dpi

Eric Meyer`s CSS

Der US-Bestseller
"More Eric Meyer on CSS"

ADDISON-WESLEY

PEARSON Education

PEARSON Education

ADDISON-WESLEY

Eric Meyers CSS

Aus dem Englischen von Jürgen Dubau

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten
sind im Internet über http://dnb.ddb.de abrufbar.

10 9 8 7 6 5 4 3 2 1

07 06 05

ISBN 3-8273-2296-0

Übersetzung: Jürgen Dubau, Freiburg/Elbe (www.dubau.net)
Satz: Ulrich Borstelmann, Dortmund (www.borstelmann.de)
Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de
Korrektorat: Dr. Claudia Nölker (www.softpearls.de)
Herstellung: Claudia Bäurle, cbaeurle@pearson.de
Mitarbeit: Hanno Denker (www.querdenker.com)
Einbandgestaltung: Marco Lindenbeck, webwo GmbH, mlindenbeck@webwo.de
Druck und Verarbeitung: Kösel Druck, Altusried/Krugzell (www.KoeselBuch.de)
Printed in Germany

Inhaltsverzeichnis

Kapitel 4 Positionieren im Hintergrund

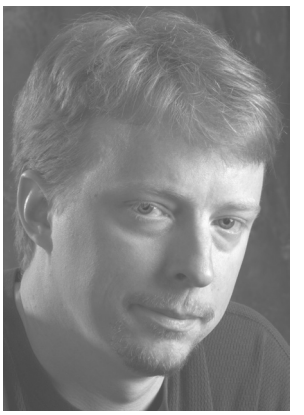
Kapitel 5 Listenbasierte Menüs

Kapitel 6 Drop-down-Menüs mit CSS

Kapitel 7 Hereinspaziert! Attraktive Tabs auf Ihrer Website

Kapitel 10 Design im Garten

Über den Autor



Eric A. Meyer arbeitet seit 1993 mit dem Web und ist für HTML, CSS und Webstandards ein international anerkannter Experte. Seine Schriften sind weit verbreitet, und seine von ihm gegründete Firma Complex Spiral Consulting (`www.complexspiral.com`) konzentriert sich darauf, den Kunden beim Geldsparen zu helfen und die Effektivität durch die Verwendung von standardorientierten Webdesigntechniken zu steigern. Macromedia und die Wells Fargo Bank zählen zu seinen Kunden.

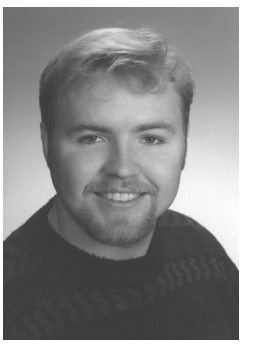
Anfang 1994 war Eric der visuelle Designer und Uni-Webkoordinator für die Website der Case Western Reserve University, wo er als Autor auch eine allgemein gelobte Serie von drei HTML-Tutorien veröffentlichte. Weiter war er Projektkoordinator der Online-Version der Encyclopedia of Cleveland History, die kombiniert mit der Dictionary of Cleveland Biography (`ech.cwru.edu`) das erste Beispiel der Enzyklopädie einer städtischen Geschichte darstellt, die vollständig und kostenlos im Web veröffentlicht wurde.

Als Autor von *Eric Meyer on CSS: Mastering the Language of Web Design* (New Riders), *Cascading Style Sheets: The Definitive Guide* (O'Reilly & Associates) und *CSS 2.0 Programmer's Reference* (Osborne/McGraw-Hill) und zahllosen weiteren Artikeln für das *O'Reilly Network*, *Web Techniques* und *Web Review* hat Eric ebenfalls die *CSS Browser Compatibility Charts* kreiert und die Erstellung und Verfassung der offiziellen CSS Test Suite des W3C koordiniert. Er hat vor einer breitgefächerten Reihe von Organisationen Vorträge gehalten, darunter das Los Alamos National Laboratory, die New York Public Library, die Cornell University und die University of Northern Iowa. Eric hat auch auf einer Vielzahl von Konferenzen technische Präsentationen und Vorträge gehalten, darunter IW3C2 WWW, Web Design World, CMP, SXSW, der Konferenzreihe User Interface und The Other Dreamweaver Conference.

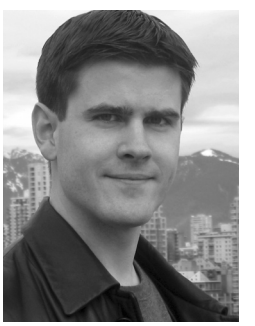
In seiner freien Zeit kümmert Eric sich als List Chaperone (»Anstandswauwau«) um die höchst aktive css-discuss-Mailingliste (`www.css-discuss.org`), die er gemeinsam mit John Allsop (Western Civilisation) gegründet hat und die jetzt von evolt.org unterstützt wird. Eric lebt in Cleveland, Ohio, die als Stadt viel netter ist, als man Sie glauben macht, und moderiert die Radiosendung »Your Father's Oldsmobile« über Bigbands, wöchentlich zu hören auf WRUW 91.1 FM in Cleveland (`www.wruw.org`). Falls er gerade nichts anderes zu tun hat, geht er auf die eine oder andere Weise seiner Frau Kat auf die Nerven.

Über die Technischen Berater

Die folgenden technischen Berater haben ihren beträchtlichen, praktischen Sachverstand bei dem gesamten Entwicklungsprozess von *More Eric Meyer on CSS* eingebracht. Als das Buch geschrieben wurde, haben sich diese Profis eingehend dem gesamten Material mit dem technischen Inhalt, der Organisation und dem Verlauf gewidmet. Ihr Feedback war absolut wesentlich, um sicherzustellen, dass *More Eric Meyer on CSS* den Ansprüchen der Leser für technische Informationen von höchster Qualität genügt.



Porter Glendinning ist der Eigentümer und Principal Consultant von Cerebellion Design. Er lebt mit seiner Frau Laura, die sich mit seiner Obsession für das Internet abgefunden hat, und dem riesengroßen gelben Labrador namens Arrow, der seine Socken frisst, vor den Toren von Washington, D.C. Porter können Sie online bei `www.g9g.org` und www.cerebellion.com finden. Er ko-administriert die Mailingliste Babble, ein Forum zur Diskussion von allen möglichen anspruchsvollen Themen des Webdesign (`www.babblelist.com`), und ist Mitglied des Web Standards Projekt Steering Committee (`www.webstandards.org`).



Dave Shea ist Schöpfer und Pfleger des höchst einflussreichen CSS Zen Garden (`http://www.csszengarden.com/`). Er ist Mitglied des Web Standards Project (`http://www.webstandards.org/`) und auch Eigentümer und Direktor von Bright Creative (`http://www.bright-creative.com`). Er schreibt in seinem täglichen Weblog `mezzoblue.com` über alle möglichen Sachen und lebt in Vancouver, B.C., Kanada. Falls Sie ihn nicht vor dem Bildschirm antreffen, ist er gewöhnlich auf den dortigen Wochenmärkten oder kleinen Kaffeeröstereien zu finden.

Danksagungen

Linda Bump Harrison und Lori Lyons, beide von New Riders Publishing/Peachpit Press, haben mir während dieses ganzen Projekts den Löwenanteil an Unterstützung und Zuspruch gegeben, und beide verdienen besondere Erwähnung für die Geduld und Toleranz, die sie bewiesen haben, als sich das Leben rücksichtslos über meine Deadlines hinwegsetzte. Was recht oft passierte.

Ganz besonders möchte ich mich bei meinen technischen Redakteuren Dave Shea und Porter Glendinning für ihre Anmerkungen bezogen auf hervorzuhebende Punkte, klärungsbedürftige Abschnitte und auszubügelnde Fehler bedanken. Ein Extradankeschön geht an Dave für das fantastische visuelle Design, mit dem er Projekt 10 verschönt hat.

Bei Douglas Bowman möchte ich mich ganz herzlich bedanken, dass er sich um das Vorwort gekümmert hat. Ich habe Dougs Arbeit immer schon bewundert, seit er 2002 Wired News neu designt hat; er hat immer wieder seine technische Finesse mit großartigem visuellen Design kombiniert, um zu wirklich ganz wundervollen Ergebnissen zu gelangen. Es ist mir eine Ehre, dass er das Buch vorstellt.

Mein weiterer Dank geht an die tausende Mitglieder von css-discuss, der Mailingliste, um die ich mich kümmere, denn sie haben diese zu einer der besten mir bekannten Ressourcen gemacht und dabei einen so hohen Anspruch gehalten. Danke auch an evolt.org und John Handelaar, die der Liste ein Zuhause gegeben haben und dafür sorgen, dass sie glatt läuft, und an John Alsopp von Western Civilisation, der mir geholfen hat, die Mailingliste überhaupt an den Start zu bringen.

Wie immer möchte ich gerne meinen tiefen Dank allen Menschen aussprechen, die mich über die Jahre mit Lob, Beschwerden, Kommentaren, Vorschlägen, Fragen und Ideen bezogen auf CSS, Browser und meine Abhandlungen versorgt haben. Es tut mir Leid, dass ich nicht allen antworten konnte, aber ich habe alles gelesen, was ihr mir geschrieben habt. Dank geht an jeden Einzelnen von euch.

Schließlich meine Frau Kathryn – du bist die tollste Gefährtin, die sich ein Mann jemals erträumen konnte. Ohne deine Unterstützung, Kraft und deinen festen Glauben hätte ich niemals so viel geschafft oder wäre so weit gekommen, und ich werde dir nie genug dafür danken können, was du für mich getan hast und was du mir bedeutest!

Eric A. Meyer

Februar 2004

Vorwort von Kai Laborenz

Ganz klar – CSS sind in der Realität des Webdevelopment angekommen. Immer mehr große Websites wagen den Sprung zum tabellenfreien Layout – post.de, stern.de, GMX oder Lycos.com sind nur einige davon.

Mittlerweile hat sich herumgesprochen, dass sich nur mit Cascading Stylesheets schlanke, barrierearme und trotzdem ansehnliche Websites gestalten lassen. Moderne Browser wie Firefox tun ein übriges, um die breite Verwendung von CSS zu ermöglichen.

Das war nicht immer so – noch vor drei Jahren schlugen sich die wenigen CSS-Webdesigner mit alten Browsern wie dem Internet Explorer 5.0 oder der berüchtigten 4er-Version von Netscape herum. Informationen oder Literatur zur praktischen Anwendung von CSS und zum Umgang mit den verschiedenen Browsern waren kaum zu finden.

Ein Name tauchte aber immer in Zusammenhang mit CSS auf – Eric Meyer.

Ich hörte zum ersten Mal von Eric, als ich 2002 – während ich mich mühsam durch die Spezifikationen von CSS wühlte – bei einem Aufenthalt in San Francisco ein Buch mit dem unscheinbaren Titel »Eric Meyer On CSS« entdeckte. Ich war auf Anhieb fasziniert – hier wurden zum ersten Mal CSS so erklärt, wie sie im Browser funktionieren – und nicht, wie es die Spezifikationen erwarten (der Unterschied war 2002 noch um einiges größer als heute ...).

Seitdem hat Eric mir vor allem mit seiner Website CSS/edge (www.meyerweb.com) und der von ihm betreuten Mailingliste css-discuss (www.css-discuss.org) in vielen Momenten die übermüdeten Augen geöffnet, Probleme gelöst und völlig neue Wege aufgezeigt. Danke, Eric!

Mit »Eric Meyer's CSS« gewährt er uns neue Einblicke in die aktuelle Entwicklung von CSS und zeigt damit den Weg zu standardkonformen und trotzdem visuell faszinierenden Websites.

Die Verwendung von semantischem HTML – beispielsweise des Listenelements <ul> als Auszeichnung für Navigationsmenüs – zeigt, wohin die Entwicklung des Webdesigns gehen wird.

Und nicht nur die des Webdesigns – Eric Meyer ist schon weiter: Auf seiner Website stellt er sein komplett auf CSS basierendes Präsentationssystem S5 (»simple standards-based slide show system«) vor, mit dem sich elektronische Präsentationen ganz ohne zusätzliche Software erstellen lassen – nach offenen Standards, lauffähig auf allen Systemen und Plattformen und frei zum Download (meyerweb.com/eric/tools/s5/).

Semantisches (X)HTML und Cascading Stylesheets sind die Werkzeuge, mit denen das Web jetzt gebaut wird – und Eric Meyer zeigt in »Eric Meyer's CSS« einmal mehr, wie man damit umgeht.

Kai Laborenz, September 2005

www.css-praxis.de

Vorwort von Douglas Bowman

»Eric Meyer.« Sprechen Sie diesen Namen aus, und Sie haben sofort meine vollste Aufmerksamkeit und können mich wahrscheinlich in ein Gespräch verwickeln, auch wenn wir einander völlig fremd sind. Vergangenes Jahr stöberte ich gerade in der technischen Abteilung eines Buchladens, als ich hörte, wie eine Frau ihrer Begleitung den Namen eines Buches nannte, das sie gerade durchgeblättert hatte: »Das hier heißt *Eric Meyer on CSS*. Ich glaube, von dem hab ich schon mal gehört.«

Ich trat ein wenig näher, bat um Entschuldigung und bot ungefragt meinen Rat an: »Wenn Sie überlegen, dieses Buch kaufen zu wollen, zögern Sie nicht!«

Als ich mich vergewissert hatte, dass sie sich wenigstens mit den Grundlagen von CSS auskannte, habe ich ihr meine Empfehlung erklärt. »Das Buch ist gut, und Eric Meyer ist gut. Wenn Sie nur ein Kapitel dieses Buches durcharbeiten, werden Ihnen ganze Kronleuchter aufgehen.«

Wir sprachen noch ein wenig über das Buch und mein Wissen über Eric Meyer. Sie dankte mir, klemmte sich das Buch überzeugt unter den Arm und machte sich auf den Weg zur Kasse.

Wenn Sie wüssten, wie entscheidend Eric Meyer dafür gewesen ist, mein Verständnis von CSS völlig umzukrempeln, und wie ich es für das Erweitern der Grenzen des Webdesigns einsetze, dann wüssten Sie auch, warum ich dieses Buch auch völlig Fremden vorbehaltlos empfehle.

Wissen Sie, ich habe CSS jahrelang ignoriert.

Als ich bei HotWired gearbeitet habe, dachten meine Kollegen, dass ich CSS liebe, und ermutigten mich bei jeder Gelegenheit, in die Welt des Webdesigns mit Stylesheets einzutauchen. Obwohl ich zuallererst ein Designer bin, wussten meine Kollegen, dass mein Verstand eindeutig technisch orientiert ist und sich mit klar umrissenen logischen Konzepten beschäftigt. Allerdings kann ich, wenn es um Code und sein Verhalten geht, Inkonsistenz und Unberechenbarkeit nicht gut tolerieren.

Als ich schließlich dem Druck nachgab und mit CSS herumzuspielen begann, fuhr ich buchstäblich gleich vor die Wand. Bei den 4.0-Versionen von Netscape Navigator und Internet Explorer war ich jedes Mal nur frustriert, wenn ich versuchte, CSS bei mehr als nur der Farbe und einfacher Schriftgestaltung einzusetzen. Ich wollte bei allen üblichen Browsern und Plattformen konsistente Ränder, Schriftgrößen und Positionierung sehen. Im Jahre 1998 war der Support auch für diese grundlegenden Features einfach grauenhaft, und das hat jedem Designer einen heftigen Brummschädel bereitet, der den gleichen Look in verschiedenen Browsern produzieren wollte.

Also tat ich CSS als schönes Wunschdenken ab, das für mich ganz gewiss keine Bedeutung hatte. Ich wollte weiterhin schönes nützliches Design produzieren, und ich hatte keine Lust, mich dabei beim Implementieren und Kontrollieren meines Designs auf CSS und dessen schlechte Browserunterstützung zu verlassen.

Während dieser Widrigkeiten mit CSS hat mich das Schicksal auf eines der wenigen Bücher gestoßen, das sich damals vollständig den Cascading Style Sheets widmete. Es war mein Glück, dass der Autor Eric Meyer hieß.

Erics Buch verstaubte ein paar Jahre unbenutzt im Regal, während ich einen großen Bogen um CSS machte. Nach und nach begannen sich die Umstände zu ändern. Ich traf auf Meldungen über deutlich verbesserten Browsersupport für CSS. Kleine Sites nutzten CSS in steigendem Maße, und es sah danach aus, als ob sie recht konsistente Ergebnisse produzierten. Diese Veränderungen haben mich so sehr gereizt, dass ich mich erneut mit CSS beschäftigen wollte und nach mehr Informationen suchte.

Bei beinahe allen hilfreichen Ressourcen traf ich auf den Namen Eric Meyer – den Autor des Buches in meinem Regal! Artikel über CSS, CSS-Testsuites, eine CSS-Mailingliste und sein *CSS Master Grid*, das ich mit religiöser Inbrunst nutzte, um mögliche Kombinationen von Eigenschaften und Werten auszuprobieren.

Sein Buch, dass ich Jahre zuvor erstanden hatte, war nun nicht mehr ins Regal verbannt, sondern lag griffbereit auf meinem Schreibtisch, und ich nutzte es nun als Nachschlagewerk regelmäßig. »Wie ging das noch mal mit dem Positionieren?« »Wie nennt man bei CSS noch mal das Tracking?« »In welcher Reihenfolge müssen diese Font-Daten angegeben werden?« Ich konnte einfach nicht genug von Eric Meyers Schriften bekommen. Jeder neue Artikel von Meyer, den ich entdeckte, hat mir geholfen, dem Puzzle ein neues Teil hinzuzufügen.

Spulen wir etwa ein Jahr vor. Erics erstes Buch lag immer noch auf meinem Schreibtisch, mittlerweile ziemlich abgegriffen. 2002 steckte ich bis zum Hals im völlig auf CSS umgestellten Redesign von Wired News, als *Eric Meyer on CSS* herauskam – sein erstes, unschätzbar wertvolles Buch, das CSS projektorientiert behandelte. Als ich das neue Buch in die Finger bekam, tauchte ich gleich am ersten Abend in das Kapitel über mehrspaltiges Layout ab. Ich hing sofort am Haken. Während ich das Buch durcharbeitete, ließen mich die vielen Erleuchtungen wünschen, es wäre erschienen, lange bevor ich die komplexen Stylesheets von Wired geschrieben hatte (und ich hätte es damals auch gelesen).

Also können Sie sich meine Aufregung vorstellen, als ich erfuhr, dass Eric bei New Riders diesen Nachfolger für *Eric Meyer on CSS* herausbringen wollte! Also kriegen wir noch mehr von einer damals schon großartigen Sache! Mehr praktische Beispiele, die für alle Webdesigner und Entwickler voll ins Schwarze treffen. Mehr praxisnahe Projekte, in denen Herausforderungen nachgebaut werden, denen wir uns täglich stellen müssen. Mehr von Erics tiefgründigen Offenbarungen und Einsichten in Grundlagen und anspruchsvolle Verwendung von CSS. Mehr weise Worte des Meisters. *More Eric Meyer on CSS!*

Erics Wissen und Können bei CSS befähigt ihn, dieses Thema maßgeblich zu behandeln. Und doch schreibt er in diesem persönlichen, familiären Ton, der so leicht zu lesen und zu verstehen ist. Diese Kombination sorgt bei einem Lehrer für das Beste beider Welten – ob Sie nun etwas ganz Neues lernen oder Ihr bisher gesammeltes Wissen erweitern wollen. Sie werden durch die

zugrundeliegenden Konzepte geführt, »wie« er etwas macht, und gleichzeitig erklärt er mühelos das »Warum«. Ich bin der Meinung, dass das Begreifen des »Warum« bei CSS genauso wichtig ist wie das »Wie«. Mit dem projektbezogenen Ansatz dieses Buches und seines Vorgängers trifft Eric genau die richtige Balance zwischen beiden.

Wenn Sie Erics Beispieldateien ansehen oder sich die Anmerkungen und Warnungen am Rande durchlesen, erscheinen in schöner Regelmäßigkeit Glühbirnen über Ihrem Kopf: »Ach, so kriegt man das hin!« oder »Oh, darum verschwindet mein Hintergrund unter dem Float! Jetzt hab ich's kapiert.»

Schlagen Sie sich mit einem ärgerlichen Problem im Layout eines Kundenprojekts herum? Versuchen Sie herauszufinden, wie Sie Ihre Bildergalerie flexibler gestalten können? Keine Ahnung, warum sich diese Hintergrundbilder nicht gerade ausrichten lassen? Widmen Sie diesem Buch ein wenig Zeit, und Eric führt Sie durch eine ähnliche Situation, die Ihnen die Augen für die verschiedenen Möglichkeiten öffnet. Später werden Sie sich fragen, was Sie bloß angestellt hätten, wenn Eric Meyers Anleitung Ihnen nicht auf die Sprünge geholfen hätte.

Wie schon bei der ersten Version von *Eric Meyer on CSS* erleichtert Ihnen die Machart dieses Nachfolgers, einfach in ein beliebiges Kapitel einzusteigen und CSS auf eine Weise besser zu verstehen, die für Sie sofort relevant ist. Die lebensnahen Herausforderungen, die Eric hier in diesem Buch präsentiert und löst, bringen Sie auf neue Ideen, und das gibt Ihnen das Zutrauen, dass Sie auch andere, ähnliche Herausforderungen direkt anpacken können.

Bei mir war es die absolute Kehrtwende – erst habe ich CSS völlig abgelehnt und dann begierig alle neuen Methoden oder Techniken angenommen, die ich in die Finger kriegen oder damit erfinden konnte. Eric Meyer hat bei dieser Kehrtwende und in meiner Würdigung für die Leistungsfähigkeit und Flexibilität von CSS ganz gewiss eine signifikante Rolle gespielt (und tut das noch immer). Das kann er auch für Sie leisten.

Wenn Sie ein Webdesigner oder Entwickler sind, der wenigstens schon mal mit CSS herumgespielt hat und über die Grundlagen hinaus ist, stellt sich aus meiner Perspektive als echte Frage nicht, ob Sie sich dieses Buch – oder den ersten *Eric Meyer on CSS* – zulegen sollten. Sondern vielmehr, mit welchem Projekt Sie beginnen werden, wenn Ihnen das Buch endlich gehört!

Douglas Bowman

Gründer und Direktor von *Stopdesign*

San Francisco

Einführung

Was Sie gerade in den Händen halten, ist mehr oder weniger eine Fortführung von *Eric Meyer on CSS*, das im Jahre 2002 mit einem sehr positivem Echo veröffentlicht wurde. Die projektbezogene Herangehensweise kam überaus gut an, und offensichtlich mochten sehr viele Leute das Gefühl, mir scheinbar über die Schulter schauen zu können, während ich mich durch die Projekte arbeite. Das war genau der Eindruck, den ich erreichen wollte, und ich werde mir Mühe geben, das auch in diesem Buch zu bieten.

Wenn Sie also dieses Buch erwerben und die Inhalte Ihnen gefallen, finden Sie noch mehr davon in *Eric Meyer on CSS*. Andererseits möchte ich darauf hinweisen, dass Sie *Eric Meyer on CSS* nicht kennen müssen, um mit diesem Buch zu arbeiten und es zu genießen. Jedes Buch für sich steht als abgeschlossenes, eigenständiges Werk da. Also brauchen Sie nicht zu befürchten, dass Sie Personen und Handlung aus diesem Buch nicht verstehen, weil Sie den Vorgänger nicht gelesen haben. Zudem gibt es in diesem Buch keine Personen.

Und doch gibt es eine Handlung (in Wahrheit sind es zwei Handlungsstränge). Der erste beschreibt eine Reise des Lernens und des Experimentierens, in dem unser Held (das sind Sie) dem Pfad eines erfahrenen Anleiters folgt und die Wege in einem neuen und wunderbaren Land kennen lernt. Der zweite Strang (so etwas wie eine Nebenhandlung) ist ein heimtückischer Versuch, Sie zur Verwendung von mehr CSS zu verführen, indem Sie mit flexiblen Designs, verbesserter Web-Accessibility, reduziertem Seitengewicht und coolen visuellen Effekten geködert werden.

Sollten Sie dieses Buch kaufen?

Das ist keine Scherzfrage. So stolz ich auch auf die Arbeit bin, die in diesen Seiten steckt, ist mir doch auch ganz deutlich klar, dass dieses Buch nicht für jeden Leser geeignet ist. Lassen Sie mich also einmal kurz zwei Arten von Lesern beschreiben: die, für die dieses Buch geschrieben wurde, und die anderen.

Für wen dieses Buch gedacht ist

Sie werden dieses Buch hilfreich finden, wenn eine oder mehrere der folgenden Kriterien auf Sie zutreffen:

- Sie wollen eine ganz praktische Anleitung zur Verwendung von CSS in echten Projekten. Das ist genau das Thema des gesamten Buches.

- Sie lernen gerne, indem Sie die Schritte selbst praktisch nachvollziehen – jemand, der viel mehr aus interaktivem Experimentieren ziehen kann als einfach nur aus der Lektüre eines Buches. Mal abgesehen von der Tatsache, dass es sich hier nun mal um ein Buch handelt, ist es tatsächlich so aufgebaut, dass der Leser die Schritte zu Hause selbst nachvollzieht.
- Sie wollten schon länger Ihre Kenntnisse von CSS verbessern, aber Sie schieben es immer vor sich her, weil CSS ein umfangreiches, komplexes Thema ist, und Sie keinen Plan haben, wie Sie auf das nächste Level kommen können.
- Sie wollten schon immer mal jemanden haben, der Ihnen zeigt, wie man typisches, herkömmliches, reines HTML-Design in echtes CSS-Design konvertiert und Ihnen erklärt, warum das für Sie von Vorteil ist. Wenn das der Fall ist, dann gehen Sie nicht über Los, sondern ohne weitere Verzögerung zum Projekt 1 »Eine existierende Seite konvertieren«.
- Wenn Sie gefragt werden, dann beschreiben Sie Ihre eigenen HTML-Kenntnisse als »mittelmäßig« oder »fortgeschritten« und Ihre eigenen CSS-Kenntnisse als »Grundkenntnisse« oder »mittelmäßig«. Anders gesagt verstehen Sie HTML ziemlich gut und haben ausreichend CSS benutzt, um eine grundlegende Vorstellung davon zu haben, wie es geschrieben wird.

Für wen dieses Buch nicht gedacht ist

Sie werden dieses Buch nicht nützlich finden, wenn eine oder mehrere der folgenden Sachen auf Sie zutreffen:

- Sie haben noch nie CSS verwendet oder sogar noch nie gesehen. Obwohl einige grundlegende Begriffe im Text definiert werden, gehen wir doch von der Annahme aus, dass der Leser die Grundlagen des Programmierens in CSS kennt und beim Erstellen von HTML ziemlich bewandert ist. Einige Leser von *Eric Meyer on CSS* haben berichtet, dass sie damit arbeiten konnten, obwohl sie vorher kaum mit CSS zu tun gehabt haben, aber dieses Buch richtet sich definitiv nicht an Anfänger.
- Sie wollten all die theoretischen Feinheiten verstehen, die in CSS stecken, und die Nuancen der Spezifikation begreifen. Es gibt heutzutage eine Menge Bücher auf dem Markt, die diese Nische besetzen. Unser Schwerpunkt hier liegt auf der Demonstration funktionierender Effekte.
- Sie haben sich mit Webdesign bisher nur mit einer grafischen Oberfläche beschäftigt. Dieses Buch geht davon aus, dass Sie HTML und CSS »freihändig« schreiben können (oder geschrieben haben), und sein Schreibstil beruht auf dieser Annahme. Die hier vorgestellten Projekte können wohl leicht in einem Mausklick-Editor reproduziert werden, aber das Buch wurde nicht mit Blick auf solche Editoren geschrieben. Es hat sich herausgestellt, dass *Eric Meyer on CSS* bei vielen Dreamweaver- und GoLive-Usern ein großer

Hit war, also sollte man das berücksichtigen. Nichtsdestotrotz geht der Text davon aus, dass Sie direkt mit dem Markup und den Styles arbeiten werden.

- Sie wollen ein Buch, dass Ihnen sagt, wie CSS geschrieben wird, das in allen Browsern und auf allen Plattformen gleich aussieht, einschließlich Netscape 4.x und Explorer 3.x. Schauen Sie sich dazu den folgenden Abschnitt »Was Sie von diesem Buch erwarten können« an.
- Sie haben meine anderen Bücher gelesen und hassen den persönlichen, familiären Stil, in dem ich schreibe. Ich kann Ihnen versichern, dass mein Schreibstil sich nur wenig geändert hat.

Was Sie von diesem Buch erwarten können

Von Anfang an war meine Absicht, ein attraktives, interaktives Buch zu schreiben, das sich auf eine praktische und interessante Verwendung von CSS konzentriert und in aktuellen Browsern einsetzbar ist. Jedes Projekt startet in seiner Entwicklung dabei von einem Anfang ohne Styles, bis es vollständig gestylt und zum Einsatz im Web bereit ist. Wenn ich meinen Job gut gemacht habe, dann sollten Sie das Gefühl haben, mir über die Schulter zu schauen, während ich an einem Projekt arbeite und das kommentiere, was ich gerade mache.

Obwohl Sie einfach den Text lesen und sich die Bilder anschauen können, um ein Gefühl für den Fortgang des Projekts zu bekommen, denke ich, dass es der beste Weg für die Arbeit mit diesem Buch sein wird, bei der Lektüre einen Webbrowser und einen Texteditor geöffnet zu haben. Auf diese Weise können Sie die Änderungen nachvollziehen, die ich im Text vornehme, indem Sie die gleichen Änderungen in Ihrer Projektdatei machen und die Änderungen in Ihrem eigenen Webbrowser sehen.

Da gibt es einen Punkt, den ich sehr deutlich betonen möchte: Die Techniken in diesem Buch sind allgemein für Browser gedacht, deren Versionsnummer größer oder gleich 5 ist (na ja, und für Safari 1.0+). Wenn Sie eine Site designen müssen, die im Explorer 4.x und Netscape 4.x genauso aussehen soll wie in IE 6.x und NS 7.x, dann ist dieses Buch wohl nichts für Sie.

Projektübersicht

Um die durchgängig praxisnahe, alltagstaugliche Natur des Buches zu bewahren, habe ich es in zehn Projekte aufgeteilt – jedes geht über genau ein Kapitel. Sie können beliebig von Projekt zu Projekt springen, wie es Ihnen in den Sinn kommt, weil jedes Projekt so geschrieben wurde, dass es so eigenständig wie möglich ist. Allerdings geht das Buch doch von einem »linearen Leser« aus; wenn Sie es von Anfang bis Ende lesen, werden Sie herausfinden, dass die Projekte aufeinander aufbauen.

Mit ein paar Ausnahmen haben die Projekte selbsterklärende Namen. Beispielsweise nimmt sich das Projekt 1 (»Eine existierende Seite wird konvertiert«) eine Seite vor, die nur HTML-Markup und Pixel-GIFs verwendet, und konvertiert sie in ein reines CSS-Design, das für das Layout keine Tabellen, sondern Positionierung verwendet.

Die Projekte 2 und 3 decken einige recht grundlegende Projekte ab: Da wird eine Fotogalerie gestaltet und gezeigt, wie ein Finanzbericht besser aussehen kann als der Standard. Projekt 4 hebt die Messlatte ein wenig höher, indem wir zeigen, wie Hintergründe auf kreative Weise genutzt werden können und wie relative und absolute Maße gemischt werden können, um Transparenz-Effekte zu erzielen.

Dann folgt in den Projekten 5 bis 7 als Kernthema die Verwendung von Listen auf verschiedene Weisen. Das erste dieser drei Projekte verwendet eine Liste von Links, um ein Menü in einer Seitenleiste zu erstellen, und zeigt zwei komplett verschiedene Layouts für die gleiche Liste. Das zweite Projekt dieser Trilogie nimmt eine Serie von verschachtelten Links und verwandelt sie in einen dynamischen Set von Drop-down-Menüs, die in den meisten Browsern funktionieren (und da ist der IE/Win mit von der Partie). Das letzte der drei Projekte erforscht die Verwendung der »Sliding Doors«-Technik, um eine Liste von Links in eine Reihe von »Tabs« umzuwandeln.

Die Projekte 8 und 9 drehen sich um das Styling eines Weblogs beziehungsweise der dazugehörigen Homepage. Projekt 10 ist das ehrgeizigste und komplexeste des ganzen Buches: Da wird ein Entwurf für den *CSS Zen Garden* genommen und die Applikation vom Design bis zum Markup durchgegangen. Dafür haben wir ein Design vom Schöpfer des *Garden* erbeten und daran gearbeitet, dieses Design in ein gestyltes Dokument zu übertragen. Für diejenigen unter Ihnen, die in der Welt des Prints arbeiten, nehmen wir ein Reinlayout und verwandeln es in ein fertiges Produkt.

Die begleitende Website

Jedes Projekt in diesem Buch basiert auf der Bearbeitung einer echten Projektdatei. Sie können die Projektdateien entweder auf einen Rutsch herunterladen oder für jedes Kapitel einzeln. Die Projektdateien bekommen Sie auf der das Buch begleitenden Website: http://more.ericmeyeroncss.com/. Dort werden Sie die Dateien finden, die wir benutzt haben, um die im Buch verwendeten Abbildungen zu produzieren, etwaige Errata für dieses Buch, und Zusatzmaterial wie Extratexte, Kommentare des Autors und Links auf nützliche Online-Ressourcen.

Für jedes Projekt finden Sie dort ein Archiv mit all den Dateien, die Sie brauchen, um mit dem Text arbeiten zu können; das schließt auch alle nötigen Grafikdateien ein sowie eine Version der Projektdatei für den Anfang. Diese Dateien folgen einem einheitlichen Namensschema: Beispielsweise trägt die Abbildung, die zu der Abbildung 7.3 gehört, den Namen `ch0703.html`, und die Arbeitsdatei für das Projekt 1 heißt `ch01proj.html`. Dies ist die Datei, die Sie im Ver-

lauf des Projekts mit einem Texteditor öffnen und darin Ihre Änderungen vornehmen sollten. Sie können sie auch in einen Webbrowser laden und nach jedem Schritt auf *Aktuelle Seite neu laden* klicken, um zu sehen, welchen Effekt die neuen Styles haben.

Sie finden dort auch die Projektdateien vom ersten Buch *Eric Meyer on CSS*. Die können Sie sich herunterladen, egal ob Sie das Buch gelesen haben oder nicht, aber das macht wohl wenig Sinn, wenn Sie das Buch nicht kennen.

Konventionen

Dieses Buch folgt einigen typographischen Konventionen, die Sie sich merken sollten, bevor Sie weitermachen.

Ein neuer Begriff wird *kursiv* dargestellt, wenn er das erste Mal vorkommt. Oft folgt dann eine kurze Erklärung des Begriffs. Programmtext, Funktionen, Variablen und andere »Computerwörter« werden in einer nicht-proportionalen Schriftart gesetzt. Im regulären Text erhalten sie auch dunkelblaue Farbe – wenn beispielsweise die Eigenschaft `margin` oder ein Wert wie `10px` erwähnt wird.

Listing-Abschnitte werden vollständig in einer nicht-proportionalen Schriftart gesetzt. Gibt es in diesen Abschnitten blauen Text, weist der auf eine Änderung des Listings im Vergleich zu vorher hin. Die meisten Listing-Abschnitte zeigen nur ein Fragment des gesamten Dokuments oder Stylesheets, und die zu ändernden oder einzufügenden Zeilen sind von unverändertem Text umgeben. Dieser Extra-Text sorgt für den Kontext und macht es Ihnen leichter, wenn Sie den Abschnitt suchen, den Sie ändern müssen, wenn Sie dem Text folgen. Hier ist ein Beispiel:

```
<head>
<title>Cleveland Eats: Matsu</title>
<style type="text/css">
/* temporäre Styles */
table {border: 2px solid red; margin: 3px;}
td {border: 1px dotted purple; padding: 2px;}
</style>
</head>
```

Jedes Computerbuch hat seinen eigenen Stil der Darstellung von Informationen. Wenn Sie es durchblättern, sehen Sie, dass dieses Buch ein interessantes Layout bietet. Hier sind die Layout-Konventionen:

Erläuterung

Hier werden detaillierte Erklärungen geboten, die sich auf den Haupttext beziehen, aber nicht selbst Teil des Projekts sind. So können alternative Ansätze oder andere Ideen als die in dem Projekt demonstrierten angeboten werden. Diese können Sie auf jeden Fall überspringen, ohne dass es den Projektfluss stört.

Hier stehen hilfreiche Anmerkungen zum Haupttext, und davon wird es eine Menge in diesem Buch geben. Sie bekommen Tipps, Kommentare, Definitionen neuer Begriffe, Randbemerkungen oder dazu gehörige Info-Häppchen.

Hier wird angezeigt, welche Punkte in einigen Browsern Probleme verursachen könnten, oder eine entsprechend ernste Warnung ausgesprochen.

Websitenotizen geben Hilfestellungen, welche Dateien Sie herunterladen oder in einem Webbrowser öffnen sollen oder welche Sachen im Web nachzuschauen sind.

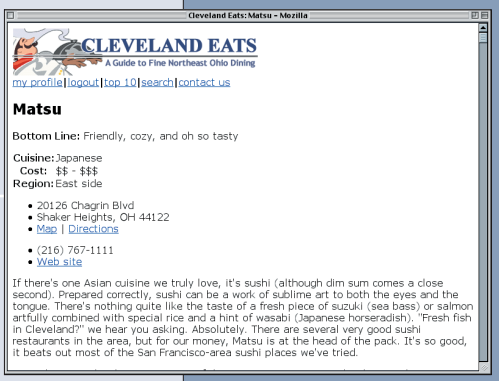
Schließlich werden Sie am Ende eines jeden Projekts einen Abschnitt namens »Spielwiese« finden. Dort werden jeweils drei kurze Übungen vorgestellt, die Sie einladen, das fertige Projekt auf verschiedene Weise zu modifizieren. Bei diesen Fortführungen sind Sie bei den Dingen, die Sie machen können, sicher nicht am Ende des Weges angelangt, aber sie können Ihnen helfen, mit den im Projekt präsentierten Konzepten zu experimentieren. Nehmen Sie sie als Ausgangspunkte für Ihre eigenen Designideen und ebenfalls als interessante Herausforderungen in eigener Sache. Wenn Sie mit Ihren eigenen Styles die Illustrationen nachbauen können, sind Sie schon auf gutem Weg, selbst kreatives CSS zu schreiben.

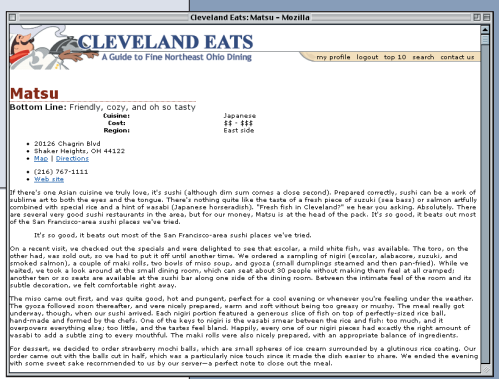
1

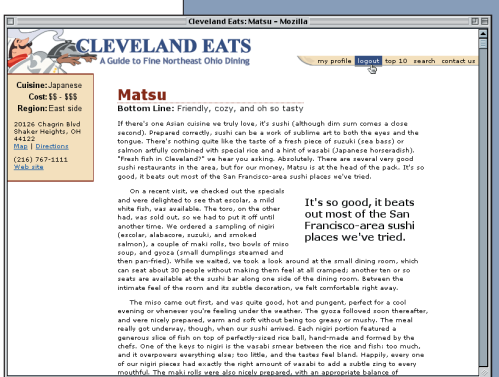
Eine existierende Seite wird konvertiert

Look inside a typical CSS [designer's] house. What do you see? Chairs, only chairs. No tables.

Jean-Yves Stervinou







Seit vielen Jahren – beim Schreiben dieses Buches sind es acht – verwenden wir Tabellen und Pixel-GIFs, um Webseiten zu entwerfen. Anfangs war das die einzige Möglichkeit, ein überzeugendes visuelles Design zu entwerfen. Tools wurden entwickelt, um diesen Bedarf zu decken, Designerschmieden nahmen das begeistert auf, und als Folge davon wurden die Seiten immer aufgeblähter.

Das Aufkommen von CSS nährte die Hoffnung, dass sich der Trend umkehren und die Seiten kleiner und aussagekräftiger werden könnten. Mit der Einführung der Positionierung unter CSS2 war der Weg frei. Es wurde theoretisch möglich, fast alles zu machen, was Tabellen machen konnten, mit einem Bruchteil der Seitengröße.

So jedenfalls die Theorie: In der Praxis sah es wenigstens für einige Jahre anders aus, was wir unvollkommenen und inkompatiblen Browser-Implementierungen zu verdanken haben. Das besserte sich nur langsam bis zum Beginn des 21. Jahrhunderts, als positionierte Layouts nur durch die weite

Verbreitung von Netscape 4.x unterbunden wurden, und sogar dort war einfaches Positionieren möglich.

Ein guter Weg, sich mit den Grundlagen des positionierten Layouts vertraut zu machen, ist, sich ein tabellenbasiertes Layout vorzunehmen und es in CSS-Positioning (CSS-P) zu konvertieren. Dies erlaubt Vergleiche zwischen den Dokumentenstrukturen und dient als Leitfaden, wie Sie sich mit einfachem Positionieren eine Menge erleichtern können.

1.1 Projektziele

Unser Ziel für dieses Projekt lässt sich leicht beschreiben: Wir nehmen ein klotziges HTML-Design und konvertieren es zur Verwendung in einem CSS-basierten Layout. Damit werden wir ausprobieren, wie allgemein übliche HTML-Strukturen und -Tricks durch ein deutlich einfacheres Markup und CSS ersetzt werden können, und wie dadurch das Dokument-Markup viel einfacher lesbar wird. Wenn wir dies abgeschlossen haben, werden wir einige Messungen vornehmen, um zu bestimmen, wieviel wir durch unsere Mühen eingespart haben.

Wir werden das Dokument abschnittsweise besprechen, also wird nicht von vornherein klar sein, welche Herangehensweise wir wählen werden. Dennoch können wir einige allgemeine Ziele formulieren:

- Die Anzahl der Bilder auf einer Seite sollte auf ein Minimum reduziert werden. Dies bringt doppelten Gewinn, da die Dokumentenstruktur deutlich sauberer wird und die für die Anzeige der Seite notwendigen Server-Hits reduziert werden.
- Alle Tabellen, die einzig und allein dem Layout dienen, sollten entfernt werden. Wenn wir fertig sind, sollten nur noch Tabellen übrig sein, in denen zu Tabellen passende Daten enthalten sind.
- Das aus unserer Konvertierung entstehende Markup sollte eine starke Struktur haben, das heißt zum Beispiel, dass Überschriften von Heading-Tags wie `h2` eingefasst sein sollen. Obendrein soll der Inhalt derart geordnet sein, dass er auch sinnvoll ist, wenn die Seite gänzlich ohne Styles angezeigt wird.
- Das Endprodukt sollte dem bisherigen reinen HTML-Dokument so nahe wie möglich kommen. Vielleicht gibt es keine hundertprozentige pixelgenaue Übereinstimmung, aber wir sollten doch unser Möglichstes tun, alle Differenzen zu minimieren.

Wenn wir alle diese Ziele erreichen, ist uns etwas recht Beachtliches gelungen.

1.2 Vorbereitungen

Laden Sie von der Homepage dieses Buches die Daten für Projekt 1 herunter. Wenn Sie vorhaben, für sich die einzelnen Schritte nachzuvollziehen, dann laden Sie bitte die Datei `ch01proj.html` in einem Editor Ihrer Wahl. Dies wird die Datei sein, die Sie im Verlauf des Projektes bearbeiten, speichern und neu laden.

Lesen Sie die Einführung für Hinweise, wie Dateien von der Website heruntergeladen werden.

1.3 Die Grundlagen werden geschaffen

Als Erstes werden wir uns nun die vorhandene, ganz mit HTML erstellte Seite in einem Web-Browser anschauen und uns dann ihre Markups vornehmen. In Abbildung 1.1 sehen Sie die Seite.

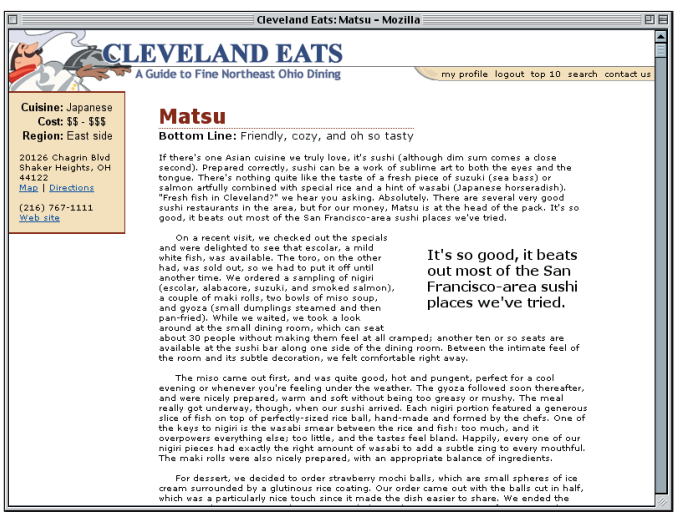


Abbildung 1.1
Die Seite in reinem HTML-Format

Nun sollten wir uns das HTML selbst vornehmen. Leider können wir das hier nicht komplett vorstellen, weil ein Listing des Quellcodes dieser Seite etwa sieben Buchseiten in Anspruch nähme. Also müssen wir anders vorgehen.

Sie können den Quellcode für die Seite aus Abbildung 1.1 sehen, wenn Sie die Datei `ch01proj.html` in Ihren gewünschten Editor laden.

Anstatt das HTML zeilenweise durchzugehen, wollen wir uns lieber anschauen, wie die Seite insgesamt konzipiert ist. Dafür werden wir dem einfachen Stylesheet, das schon am Anfang der Seite vorhanden ist, einige temporäre Styles hinzufügen.

```
<head>
<title>Cleveland Eats: Matsu</title>
<style type="text/css">
body {font-family: Verdana, Arial, Helvetica, sans-serif;}
a.navlink {text-decoration: none;}
/* temporary styles */
table {border: 2px solid red; margin: 2px;}
td {border: 1px dotted purple; padding: 1px;}
```

```
/* end temporary styles */
</style>
</head>
```

Was ist eine Regel?

Bei CSS ist eine *Regel* ein vollständiges Style-Statement, das aus einem Selektor und einem Deklarationsblock besteht. Eine solche Regel ist beispielsweise

```
p {color: gray;
font-weight: bold;
```

Die erste der neuen Regeln wird einen roten Pixelrand um alle äußeren `table`-Elemente legen und zwei Pixel Leerraum (Rand) hinzufügen.

Die zweite neue Regel gibt `td`-Elementen eine gepunktete purpurfarbene Grenze von einem Pixel und einem Pixel Zellabstand. Das Ergebnis dieser Modifizierung sehen Sie in Abbildung 1.2.

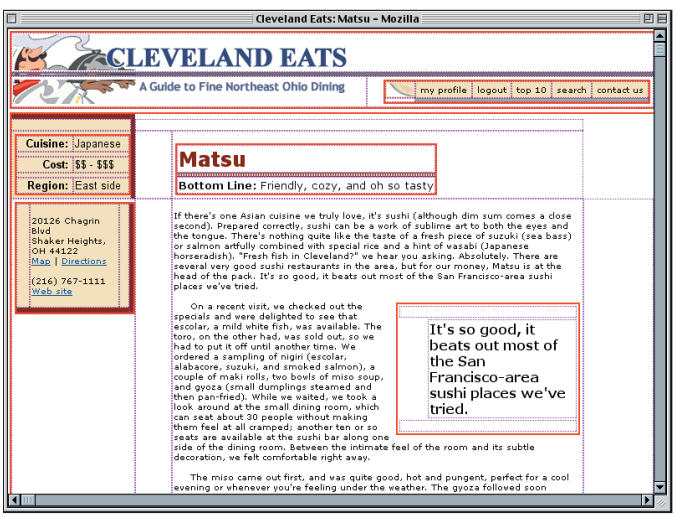


ABBILDUNG 1.2

Hinzufügen eines einfachen Stylesheets zur Verdeutlichung der Seitenstruktur

Weil jeder dicke rote Rand den äußeren Rand einer Tabelle darstellt, können wir schnell die Struktur der Seite bestimmen. Der »Banner« oder »Masthead« (»Seitenkopf«) oben auf der Seite steht in seiner eigenen Tabelle. Der Rest des Dokuments wird in eine zweite Tabelle gepackt, einige weitere Tabellen sind verschachtelt darin und ergeben eine ziemlich komplizierte Zellenstruktur. Ein interessanter Aspekt ist, dass die Seitenleiste links mit zwei getrennten Tabellen in unterschiedlichen Zellen realisiert ist, obwohl man visuell nur den Eindruck einer einzigen Box hat.

Ein anderer Layout-Trick, der von diesen temporären Styles aufgedeckt wird, ist die Verwendung von »leeren« Zellen, um Raum frei zu halten. Schauen Sie sich beispielsweise beide Seiten neben der Haupttextspalte in der Mitte der Seite an und ebenfalls auf jeder Seite des großen Textblocks, der in den Hauptinhalt gesetzt ist. (Dies wird auch als »Pullquote« («Texteinschub«) bezeichnet.) In beiden Fällen sehen Sie leere Zellen, die von einem gepunkteten purpurfarbenen Rand umrissen sind.

Diese Zellen sind nicht wirklich leer – sie enthalten Bilddateien namens `spacer.gif`. Das ist ein 1x1-Bild, dessen einziges Pixel transparent ist. Eine schnelle Suche im HTML-Code zeigt, dass es dort 18 `img`-Elemente gibt, die sich auf diese gleiche Bilddatei beziehen. Wenn wir fertig sind, werden sie alle weg sein.

1.4 Das Dokument wird konvertiert

Von einem reinen HTML-Design zu einem CSS-Layout zu kommen, erfordert zwei Schritte: Wir müssen die HTML-basierte Präsentation loswerden und in CSS neu aufbauen, um sie zu ersetzen. In vielen Fällen ist es leichter, dies nach und nach zu machen, indem man einen kleinen Teil des Dokuments auseinander nimmt und ihn stylt, bevor man sich an den nächsten Abschnitt macht. Für dieses Projekt jedoch werden wir uns den steinigen Weg vornehmen und die Datei bis ganz zum Fundament demontieren, bevor wir mit dem Styling beginnen. (Die Abbildungen sehen obendrein deutlich besser aus.)

Sie können sich die komplett abgespeckte Version der Seite anschauen, wenn Sie die Datei `ch0104.html` in Ihren bevorzugten HTML- oder Texteditor laden.

1.4.1 Abspecken bis zum Minimum

Nun wird es Zeit, eine Kopie der ursprünglichen reinen HTML-Datei zu machen und darin alle HTML-basierte Präsentation zu entfernen. Das ist ganz eindeutig der unangenehmste Teil, wenn man ein reines HTML-Design zu einem CSS-Layout konvertiert. Einen Großteil der Arbeit kann man mit einen Programm zum Suchen und Ersetzen erledigen, aber trotzdem bleibt noch die Notwendigkeit, alles durchzugehen, um übrig gebliebenes HTML manuell zu entfernen.

Das Markup aufräumen

Sie können eines von verschiedenen Utilities verwenden, um das Markup aufzuräumen und dabei die meisten oder alle Präsentationsaspekte eines Dokuments hinauszuwerfen. HTML Tidy ist so ein sehr beliebtes Tool, und man kann es kostenlos unter `http://tidy.sourceforge.net/` herunterladen.

Es soll nicht unerwähnt bleiben, dass es manchmal leichter ist, einfach den Text von einer Seite zu kopieren und eine neue Struktur drum herum zu schaffen, anstatt zu versuchen, das alte Markup zu einem neuen, effizienteren Markup zu verwandeln. Wir werden in diesem Projekt durch einen Konvertierungsprozess gehen – das wird dabei helfen zu illustrieren, wie bestimmte Arten von tabellenorientiertem Markup drastisch vereinfacht werden können. Behalten Sie nur im Hinterkopf, dass es manchmal einfacher ist, sich den Inhalt vorzunehmen und dann um diesen Inhalt herum ein neues Markup aufzubauen.

Egal wie Sie nun das Markup verschlanken, auf jeden Fall müssen folgende Dinge weg:

- `font`-Elemente
- `<br>`-Elemente
- alle Vorkommen von ` `
- alle Attribute für das Element `body` (beispielsweise `text` und `link`)

Wir wollen auch soviel Layout-Tabellen wie möglich hinauswerfen. Weil das ursprüngliche Markup auf gewisse Weise kompliziert ist, werden wir uns einen Schritt nach dem anderen vornehmen.

1.4.2 Die reine Struktur

Wir gehen nun davon aus, dass wir schon alle in der obigen Liste erwähnten Dinge herausgenommen haben. Danach bleibt ein Dokument übrig, dessen allgemeine Struktur immer noch recht schwer zu beschreiben ist. Beispielsweise enthält der Seitenkopf drei verschiedene Teilstücke eines Bildes, jedes in seiner eigenen Zelle, und dann eine andere Zelle, die eine Tabelle enthält, in der die Links der Navigationsleiste auf der Seite rechts oben stehen. Im Hauptbereich der Seite haben wir eine linksbündige Seitenleiste, die zu Ausrichtungszwecken über zwei Zeilen umgebrochen ist: Der Name des Restaurants und der »Untertitel« sind am unteren Rand der linken Tabelle ausgerichtet. Das wird über einen ganzen Wald von ineinander verschachtelten Tabellenzellen erreicht, deren Zweck ist, die Seitenleiste wie eine einzige fortlaufende Box erscheinen zu lassen, obwohl das nicht der Fall ist.

Die Liste könnte noch ein gutes Stück weitergehen, aber wir wollen jetzt einfach soviel Tabellen wie möglich rauswerfen und dann sehen, was übrig bleibt. Nachdem wir das erledigt haben, bleibt eine Dokumentstruktur wie das in Listing 1.1 gezeigte Skelett übrig.

Kursive Einfügungen

Der kursive Text im Listing 1.1 steht für Text- und HTML-Abschnitte, die für dieses Buch zu lang sind.

Listing 1.1 Der Beginn der Tabelle

```
<table>
 [...Seitenkopf...]
</table>
<div id="navbar">
 [...Links der Navigationsleiste...]
</div>
<div id="review">
 [...Titel...]
 [..."bottom line"...]
<div id="info">[...Info Seitenleiste...]</div>
 [...Restaurantkritik...]
</div>
```

Moment mal – eine Tabelle? Genau, wir werden fürs Erste die Seitenkopf-Tabelle beibehalten. Wir haben bloß die Tabelle ein wenig vereinfacht und die Navigationsleiste herausgenommen. So sieht nun das Markup für den Seitenkopf aus:

```
<table cellspacing="0" id="masthead">
<tr>
<td><img src="mast1.gif" alt="Cleveland Eats"></td>
</tr>
<tr>
<td bgcolor="#666666"><img src="mast2.gif" alt=""></td>
```

```
</tr>
<tr>
<td><img src="mast3.gif" alt="A Guide to Fine Northeast
 Ohio Dining"></td>
</tr>
</table>
```

Wir belassen es dort, weil das für den Moment einfacher ist. Das liegt einfach nur an der herausgezogenen Linie in der Mitte des Seitenkopfs, die aus der Unterseite der Servierplatte entspringt. In der Tabelle geschieht das über einen 1-Pixel-Schnitt aus dem Bild und einer Hintergrundfarbe der Zelle. Wir könnten auch das zu einer tabellenlosen Struktur konvertieren, und genau das werden wir am Ende des Projekts machen. Wenn wir mit dieser Änderung bis zum Ende warten, können wir damit jedoch ein oder zwei Sachen illustrieren.

Bis dahin schauen wir uns die Änderungen an der Navigationsleiste an. Jetzt, wo die Tabellenzellen alle weg sind, müssen wir etwas einfügen, um die Links voneinander zu trennen. In diesem Fall haben wir uns dafür entschieden, einen in ein `b`-Element eingepackten vertikalen Balken (|) einzufügen.

```
<div id="navbar">
<a
  href="/myprofile">my profile</a><b>|</b><a
  href="/logout">logout</a><b>|</b><a
  href="/top10">top 10</a><b>|</b><a
  href="/search">search</a><b>|</b><a
  href="contact">contact us</a>
</div>
```

Tabellen? Elemente in Fett? Ist das hier immer noch ein CSS-Buch? Ja! Nur Geduld. Alles wird im Laufe der Zeit schlüssig werden, aber für jetzt denken Sie daran, dass `b` ein Teil des Dokumenttyps ist, den wir verwenden (HTML 4.01 Transitional) und darum absolut gültig ist. Darauf werden wir gleich zurück kommen.

Der Rest des Dokuments ist nur ein `div`, das die gesamte Restaurantkritik einschließt, die zu einer einfacheren Struktur konvertiert worden ist.

```
<div id="review">
 <h2>Matsu</h2>
 <p id="summary"><strong>Bottom line:</strong>
    Friendly, cozy, and oh so tasty</p>
  <div id="info">[...alle Infos aus der Seitenleiste...]</div>
[...Text der Restaurantkritik...]
</div>
```

Die letzte bemerkenswerte Änderung ist, dass der Texteinschub (Pullquote) jetzt durch ein `blockquote`-Element mit einer Klassenangabe dargestellt ist.

```
<blockquote class="pull">
It's so good, it beats out most of the San Francisco-area
sushi places we've tried.
</blockquote>
```

Das war's, und das Ergebnis können wir in Abbildung 1.3 betrachten. Das sieht jetzt ein bisschen wie ein Zugunfall aus, aber warten Sie nur ab. Das wird schon gleich viel besser.

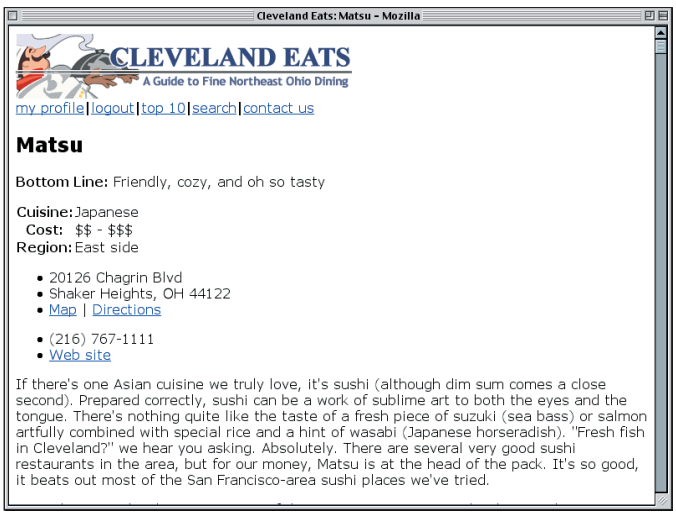


ABBILDUNG 1.3

Strukturell ist es nun besser, aber der visuelle Eindruck könnte noch etwas Arbeit vertragen.

1.5 Das Design wird neu aufgebaut

Weil es unser Ziel ist, mit CSS-Styles sowohl die Inhalte als auch das Layout des ursprünglichen Designs nachzubauen, werden wir uns für den Rest des Projekts auf das tabellenbasierte Design beziehen. Obwohl es uns nicht um eine pixelgenaue Reproduktion geht, werden wir doch unser Bestes tun, um dem Original so nahe wie möglich zu kommen. Dazu fangen wir mit ein paar grundlegenden Styles an und arbeiten uns dann durch das Dokument, um alle Teile der Reihe nach zu stylen.

1.5.1 Grundlegende Styles

Bevor wir damit beginnen, das Gesamtlayout neu zu konstruieren, wollen wir einige »globale« Styles einrichten – also solche, die sich auf das Dokument insgesamt auswirken. Als Erstes werden wir die Schriftgröße wieder so einrichten, wie sie im ursprünglichen HTML-Design aussah. Beim HTML wurden die meisten Schriftgrößen über `<font size="-2">` eingestellt, und das CSS-Äquivalent ist das Schlüsselwort `x-small`. Also werden wir die Eigenschaft `font-family` auf `font` ändern und den Größenwert einsetzen.

```
<style type="text/css">
body {font: x-small Verdana, Arial, Helvetica, sans-serif;}
a.navlink {text-decoration: none;}
</style>
```

Extra klein

Weil die originale HTML-Darstellung bei dem Body-Text eine Schriftgröße von `-2` erforderte, verwenden wir `x-small`, was auf der Liste der absoluten Schlüsselwörter zwei Stufen kleiner ist als `medium`. Achten Sie darauf, dass IE 5 das falsch versteht und `x-small` so behandelt wie eine Schrift der Größe `-1`. In IE 6 ist dieser Bug behoben.

Als Teil unseres Konvertierungsprozesses haben wir Attribute des `body`-Elements wie `marginheight` und `topmargin` hinausgeworfen. Diese brauchten wir, um den »Gutter« (die »Fugen« zwischen den Spalten) um den Dokumenteninhalt zu entfernen. Um den gleichen Effekt in CSS zu bekommen, müssen wir sowohl den Rand und das Padding von `body` auf Null setzen.

```
<style type="text/css">
body {font: x-small Verdana, Arial, Helvetica, sans-serif;
  margin: 0; padding: 0;}
a.navlink {text-decoration: none;}
</style>
```

Wir haben sowohl den `margin` als auch das `padding` gestylt, weil einige Browser den »Gutter« mit ersterem erzwingen und andere Browser mit letzterem. Wenn wir beide einstellen, haben wir alles abgedeckt – das Ergebnis sehen Sie in Abbildung 1.4.

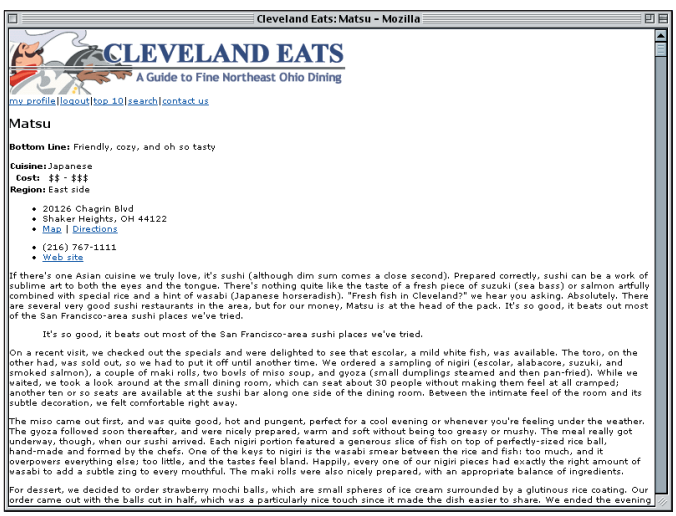


ABBILDUNG 1.4
Der Start mit ein paar globalen Styles

1.5.2 Oben herum gestrafft

Wir benutzen für den Seitenkopf immer noch eine Tabelle, und weil wir die Attribute `cellpadding` und `width` herausgenommen haben, müssen wir sie in CSS neu erstellen. Wir gehen davon aus, dass keine Tabellenzelle ein Padding haben soll und dass alle Tabellen eine Breite von 100 % erhalten. Wenn wir später eine dieser beiden Sachen wieder aushebeln müssen, werden wir das tun.

```
body {font: x-small Verdana, Arial, Helvetica, sans-serif;
  margin: 0; padding: 0;}
table {width: 100%;}
th, td {padding: 0;}
a.navlink {text-decoration: none;}
```

Jetzt machen wir uns an die Navigationsleiste. Das Markup der Links in der Navigationsleiste war anfangs wie folgt:

```
<td nowrap><font size="-2"> <a href="/myprofile"
  class="navlink"><font color="#000000">my
profile</font></a> </font></td>
```

Dies hat einen Zeilenumbruch innerhalb der Zelle verhindert, die Schriftgröße sowohl für den Link als auch den Whitespace drum herum reduziert und die Textfarbe des Links auf Schwarz gesetzt. Zusätzlich war jedem Link der Klasse `navlink` zugeordnet, was es möglich gemacht hat, die Unterstreichungen mit der Regel `a.navlink {text-decoration: none;}` zu entfernen.

Wir haben diese Klassen bei der Markup-Konvertierung herausgenommen, weil sie nicht nötig sind. Jetzt, wo die Links in einem `div` mit dem Individualformat `navbar` eingeschlossen sind, können wir einfach den Selektor der Regel ändern, mit dem die Links angesprochen werden.

```
th, td {padding: 0;}
#navbar a {text-decoration: none;}
</style>
```

Jetzt brauchen wir nur noch die Farbe der Links einzustellen und einen Zeilenumbruch in der Navigationsleiste zu verhindern. Schauen Sie sich unseren bisherigen Fortschritt in der Abbildung 1.5 an.

```
th, td {padding: 0;}
#navbar {white-space: nowrap; background: #F0DFB4;}
#navbar a {text-decoration: none; color: #000;}
```

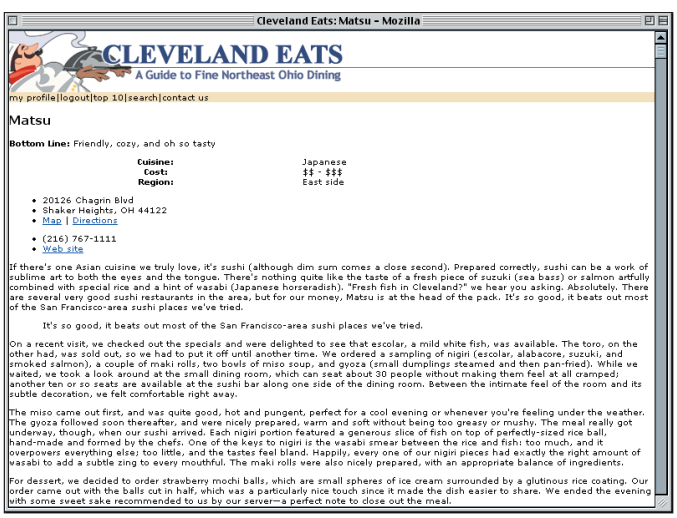


ABBILDUNG 1.5

Der Seitenkopf ist in Ordnung, nun wird die Navigationsleiste gestylt.

1.5.3 Die Navigationsleiste wird ausgefüllt

Jetzt wird es Zeit, die Navigationsleiste dort hinzusetzen, wo sie hingehört. Für diese spezielle Aufgabe werden wir das `div`-Element positionieren. Weil keines der Vorgängerelemente der Navigationsleiste positioniert ist, wird die Navigationsleiste unter Berücksichtigung des ersten Container-Blocks positioniert, was in HTML im Allgemeinen durch das `html`-Element definiert wird.

Wir wissen, dass sich die Navigationsleiste an den rechten Rand des Dokuments fügt, also brauchen wir nur noch herausfinden, wie weit nach unten wir sie platzieren müssen. Durch Auszählen finden wir heraus, dass das erste Pixel 44 Pixel vom oberen Dokumentrand unter der Trennungslinie beginnt. Also:

```
#navbar {position: absolute; top: 44px; right: 0;
  white-space: nowrap; background: #F0DFB4;}
```

Jetzt müssen wir das kurvenförmige Bild wieder an Ort und Stelle bringen. Im Original-Design war es über ein `img`-Element in eine Tabellenzelle eingeschoben, aber das haben wir beim Konvertieren des Markups herausgenommen. Wir können den visuellen Effekt reproduzieren, indem wir das gleiche Bild in den Hintergrund der Navigationsleiste platzieren.

```
#navbar {position: absolute; top: 44px; right: 0;
  white-space: nowrap;
  background: #F0DFB4 url(tab-curve.gif) bottom left no-repeat;}
```

Das hört sich nach einer prima Idee an, bis Sie erkennen, dass das Bild unter den Links platziert werden wird und nicht links davon. Um diese Art von Überlappung zu verhindern, brauchen wir ein bisschen Padding. Das Bild hat eine Breite von 32 Pixel, also werden wir fürs Erste genau diesen Padding-Wert einstellen (siehe Abbildung 1.6).

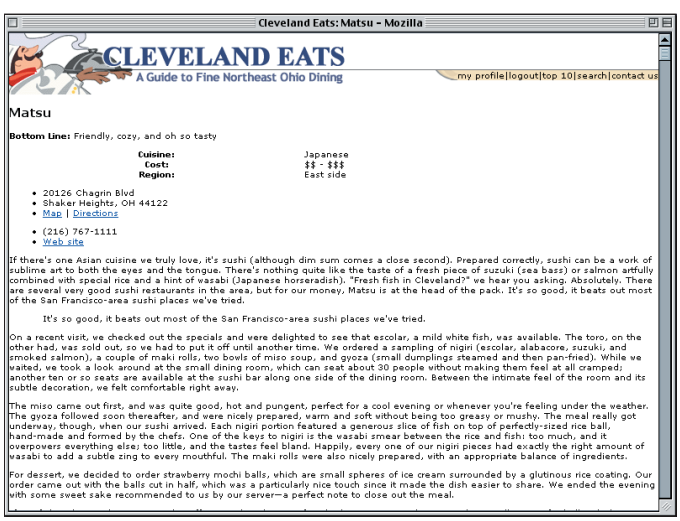


Abbildung 1.6

Die Navigationsleiste wird positioniert und ein Bild dem Hintergrund hinzugefügt.

```
#navbar {position: absolute; top: 44px; right: 0;
  padding: 0 0 0 32px; white-space: nowrap;
  background: #F0DFB4 url(tab-curve.gif) bottom left no-repeat;}
```

Das sieht schon langsam richtig gut aus, obwohl der Text immer noch ziemlich eingezwängt ist. Wir werden oben und unten noch mehr Padding einfügen, um die Geschichte vertikal etwas auseinander zu ziehen.

```
#navbar {position: absolute; top: 44px; right: 0;
  padding: 2px 0 2px 32px; white-space: nowrap;
  background: #F0DFB4 url(tab-curve.gif) bottom left no-repeat;}
```

Jetzt sollten wir uns um die vertikalen Balken kümmern. Weil wir nicht wollen, dass sie erscheinen, können wir die `boldface`-Elemente nutzen, damit sie vollständig verschwinden.

```
#navbar {position: absolute; top: 44px; right: 0;
  padding: 2px 0 2px 32px; white-space: nowrap;
  background: #F0DFB4 url(tab-curve.gif) bottom left no-repeat;}
#navbar b {display: none;}
#navbar a {text-decoration: none; color: #000;}
```

Damit werden die Elemente in einem gestylten Dokument nicht erscheinen. Wenn ein Browser das CSS nicht versteht oder die Seite mit einem nicht für CSS ausgelegten Browser aufgerufen wird, werden die Zeichen immer noch da sein, um die Links getrennt zu halten.

Das heißt natürlich, dass in einem CSS-fähigen Browser die Links direkt aufeinander folgen. Wir können sie entweder mit Rändern oder mit Padding auseinander schieben, und wir entscheiden uns für Padding. (Warum – das sehen wir gleich.)

```
#navbar a {text-decoration: none; color: #000;
  padding: 0 1em 0 0;}
```

Jetzt brauchen wir nur noch den Rahmen am unteren Rand der Navigationsleiste einzufügen. Hier wird es nun ein wenig vertrackt, weil wir dem `div` nicht einfach nur einen unteren Rahmen hinzufügen können. Wenn wir das machen würden, dann würde der Rahmen unter dem Hintergrundbild hervorschauen, und es gibt keine Möglichkeit, ein Hintergrundbild dazu zu bringen, im gleichen Element den Rahmen zu überlappen. Also bekommen statt dessen die Links selbst den unteren Rand.

```
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 0 1em 0 0;}
```

Darum haben wir die Links mit Padding auseinander geschoben. Hätten wir die Randeigenschaft verändert, dann hätten sich die Rahmenlinien nicht berührt und somit die Illusion eines einzelnen Rahmens zunichte gemacht.

Inline-Padding
Wenn die Links ein Padding am unteren Rand erhalten, ändert das in CSS-konformen Browsern im Prinzip nichts an der Höhe vom `div`. Das liegt daran, dass das Padding auf nichtersetzten Inline-Elementen (und das sind die Links) die Höhenberechnung nicht verändert. Mit ausreichendem Padding könnten wir sogar die Rahmen der Links aus dem `div` hinausschieben – zumindest bei anderen Browsern als dem Explorer.

Weil das `div`-Element nun zwei Pixel Padding für den unteren Rand hat, werden die Rahmenlinien, die wir gerade den Links hinzugefügt haben, einen Pixel über dem unteren Rand des ockerfarbenen Hintergrundbereichs sitzen. Wir müssen sie also mit ein wenig Padding hinunterschieben.

```
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 0 1em 1px 0;}
```

Damit werden nun die Rahmen am unteren Rand des Hintergrundbildes ausgerichtet, und das hat das Tabellenlayout nicht so ganz hingekriegt. Wenn wir die Rahmenlinien so weit hätten herunterschieben wollen, dass sie genau dem Effekt aus dem ursprünglichen Design entsprechen, hätten wir nur den Wert `1px` auf `2px` erhöhen brauchen, aber wir lassen alles so, wie es in Abbildung 1.7 zu sehen ist.

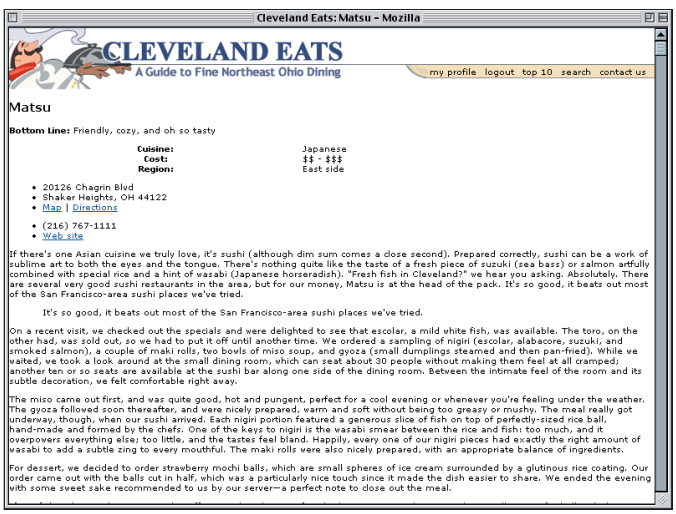


ABBILDUNG 1.7

Die Navigationsleiste wird vervollständigt.

Der große Vorteil, die Navigationsleiste zu positionieren, liegt darin, dass sie leicht verschiebbar ist. Angenommen, wir wollen sie über die Trennlinie statt darunter stellen. Im Prinzip müssten wir dann einfach nur den Wert von `top` entsprechend anpassen und das Hintergrundbild ändern. Dass nun die Navigationsleiste direkt auf der Trennlinie aufsetzt, ist durch das Positionieren ganz leicht zu schaffen. Diese Art Flexibilität macht das Positionieren als Layout-Tool so überzeugend.

Padding und Inline-Elemente

Vielleicht ist es nicht gleich offensichtlich, dass die Erhöhung des Paddings der Links dazu führt, dass sie aus dem sie umschließenden `div` *»hervorlugen«. Aufgrund der Art, wie CSS mit dem Zeilenlayout umgeht, sollte genau das passieren. Wie bei CSS festgelegt, hat ein Padding oben und unten auf nicht-ersetzten Inline-Elementen (wie Hyperlinks) keine Auswirkungen auf die Zeilenhöhe. Darum wird es, auch wenn das Padding vielleicht sichtbar ist, eine Textzeile nicht höher machen und darum sich auch nicht auf die Höhe von darin enthaltenen Elementen wie einem* `div` *auswirken.*

Andererseits sagt CSS auch, dass Browser bei Inline-Elementen das obere und untere Padding nicht zu rendern haben, und das passiert nun gerade im IE/Win. In diesem Browser werden Rahmen und Hintergrund durch die Kante des `div`*s gestutzt. Darum müssten Sie, um den gleichen Effekt zu bekommen, das Padding des* `div`*s erhöhen. Beim Rest des Projekts werden wir davon ausgehen, dass dies nicht nötig ist, aber Sie sollten das im Hinterkopf behalten, falls es mal Probleme gibt.*

Diejenigen unter Ihnen, die sich mit dem IE/Win durch das Projekt arbeiten, sollten versuchen, dem `div` *der Navigationsleiste ein oder zwei Pixel Padding oben und unten hinzuzufügen, und schauen, ob das besser aussieht.*

1.5.4 Titel und Untertitel-Styles

Nachdem unser Seitenkopf nun vollständig ist (zumindest bis wir zurückgehen und die Tabelle entfernen), wollen wir uns den oberen Bereich der Restaurantkritik selbst vornehmen. Hier finden wir ganz einfaches Markup.

```
<h2>Matsu</h2>
<p id="summary"><strong>Bottom Line:</strong>
Friendly, cozy, and oh so tasty</p>
```

Extra groß

Weil die originale HTML-Darstellung bei dem Body-Text eine Schriftgröße von `+2` erforderte, verwenden wir `x-small`, was zwei Stufen größer ist als `medium` auf der Liste der absoluten Schlüsselwörter.

Es ist ganz einfach, das Erscheinungsbild des Titels dieser Restaurantkritik nachzubauen; dafür brauchen wir bloß Größe und Farbe, um das originale Design nachzubilden.

```
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 0 1em 1px 0;}
#review h2 {color: #600; font-size: x-large;}
</style>
```

Wir müssen auch die Leerräume (oder deren Fehlen) über und unter dem Titel nachbauen. Im ursprünglichen Design gab es ein paar GIFs Zwischenraum über dem Titel, aber darunter war nur eine Zelle mit einem Hintergrund aus einer gepunkteten Linie. Anstatt den Weg über Pixel zu nehmen, werden wir es nun mit einigen em-basierten Werten probieren und später noch einmal darauf zurückkommen, um nötige Anpassungen vorzunehmen.

```
#review h2 {color: #600; font-size: x-large;
  margin: 1em 0 0;}
```

Jetzt zu der Zeile mit dem Untertitel: Diese kurze Textzeile war ein wenig größer als der Haupttext der Kritik, also werden wir deren Schriftgröße ändern müssen. Es gab da auch eine gepunktete Linie, die den Untertitel vom Titel getrennt hat, die werden wir auch einbauen. Wir können den bisherigen Fortschritt in Abbildung 1.8 anschauen.

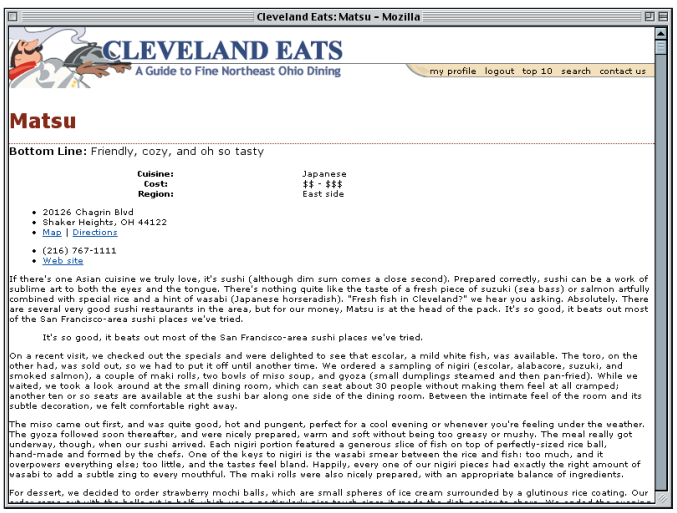


Abbildung 1.8

Titel und Untertitel kommen ins Bild.

```
#review h2 {color: #600; font-size: x-large;
  margin: 1em 0 0;}
#review #summary {font-size: small; border-top: 1px dotted #600;}
</style>
```

Hier tauchen zwei Probleme auf. Das erste ist, dass es einen Zwischenraum zwischen der gepunkteten Linie und dem Titel gibt, das ist der obere Rand des Absatzes mit dem Untertitel. Das andere Problem ist nicht notwendigerweise schlecht: Die obere Rahmenlinie geht einmal quer über die Seite. Im ursprünglichen Design war sie nur auf den Untertitel begrenzt.

Wenn das Layout erst einmal ganz fertiggestellt ist, finden wir es ja vielleicht auch ganz gut, wenn ein Rahmen von einer Seite der Hauptspalte bis zur anderen läuft. Um dem Original doch so nahe wie möglich zu kommen, lassen Sie uns eine andere Herangehensweise ausprobieren, mit der wir diesen oberen Rand loswerden und die Rahmenlinie auf die Textbreite des Untertitels begrenzen.

```
#review #summary {font-size: small; border-top: 1px dotted #600;
  display: inline;}
```

Damit haben wir die Art geändert, wie der Absatz angelegt wurde. Jetzt wird eine Inline-Box generiert, genau wie ein Hyperlink oder ein `span`-Element das täte. Der

Absatz selbst ist immer noch ein Element auf Block-Level, aber auf dem Bildschirm wird damit eine Inline-Box generiert. Also wird die Rahmenlinie am oberen Rand des Textes verlaufen, egal wie lang oder kurz der sein mag.

Dieser Lösungsweg birgt potenzielle Nachteile. Wenn der Text des Untertitels in eine zweite Zeile geht, wird diese zweite Zeile ebenfalls eine gepunktete obere Rahmenlinie bekommen. Es gibt keinen Weg, nur der ersten Zeile eines Inline-Elements einen Rahmen zuzuweisen. Wenn also die Möglichkeit besteht, dass der Untertitel in zwei Zeilen umgebrochen wird, ist das wohl kein guter Ansatz.

Der andere mögliche Nachteil wirkt sich in dem Fall aus, dass der Titel tatsächlich länger ist als der Untertitel. In diesem Fall wird der Rahmen schon vor dem Ende des Titels aufhören. Ein langer Restaurantname mit einer Unterzeile, in der »Lecker!« steht, wäre ein Beispiel dafür. Es gibt keinen einfachen Weg, das ohne Tabelle zu umgehen. Tabellen sind in der HTML-Welt insofern einzigartig, dass sich die Breite eines Elements auf die Breite verschiedener anderer auswirkt.

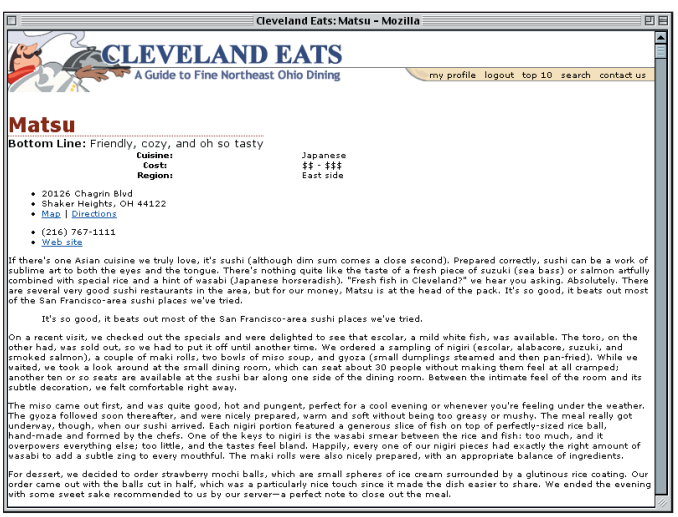
Nun sind Sie vorgewarnt und schauen sich bitte die Abbildung 1.9 an, die zeigt, dass – in diesem speziellen Fall – die Verwendung von `inline` ganz prima Ergebnisse liefert. Die Trennung zwischen dem Titel und dem Untertitel ist verschwunden, weil obere und untere Ränder bei Inline-Elementen keinen Effekt auf das Layout haben.

Linienausdehnung

Es ist allerdings möglich, den Effekt nachzuahmen, dass ein trennender Rahmen so lang ist wie die Länge von zwei Elementen, indem man negative Werte für `margin` verwendet, obwohl das recht vertrackt werden kann. Diese Technik wird in Projekt 8 vorgestellt.

ABBILDUNG 1.9

Die Zusammenfassung generiert eine Inline-Box.



1.5.5 Informationen am Rande

Während wir uns durch das Dokument arbeiten, kommen wir zum »Info«-`div`. Dies war im Original-Design als Kasten am linken Rand angelegt, und wir werden es hier genau so nachbauen. Der Vorteil liegt darin, dass vorher die Seitenleiste über zwei Zeilen geteilt und aus einer komplexen Serie von Zellen konstruiert war, während wir jetzt nur ein einziges `div` stylen müssen.

Um die Seitenleiste an Ort und Stelle zu bringen, werden wir sie positionieren. Weil keines der Vorfahrelemente der Seitenleiste positioniert ist, wird sie (wie die Navigationsleiste) unter Beachtung des ersten Container-Blocks positioniert. Also arbeiten wir dieses Mal von der linken oberen Ecke des Dokuments aus. Das Seitenkopf-Bild ist insgesamt 72 Pixel hoch, also brauchen wir die Seitenleiste nur genau unter diesen Punkt zu bringen.

```
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 0 1em 1px 0;}
#info {position: absolute; top: 73px; left: 0;}
#review h2 {color: #600; font-size: x-large;
  margin: 1em 0 0;}
```

Diese kleine Änderung bedeutet, dass das »Info«-`div` komplett aus dem Dokumentfluss herausgenommen und wunschgemäß 73 Pixel von oberen Rand des ersten Container-Blocks und genau an seiner linken Kante platziert wird. Das bedeutet, dass die Seitenleiste nun den Haupttext der Restaurantkritik vollständig überlappen wird. Tatsächlich hat die Seitenleiste die Standardbreite von 100 %, was bedeutet, dass wenn wir ihr einen Hintergrund zuweisen würden, dann würde sie sich von einer Seite des Dokuments bis zur anderen erstrecken.

Das ist natürlich noch nicht der Weisheit letzter Schluss, aber so sähe die Seite aus, wenn Sie nun deren Voransicht aufrufen. Nun wollen wir für die Seitenleiste einen Hintergrund einfügen und eine explizite Breite festlegen (siehe Abbildung 1.10).

```
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4;}
```

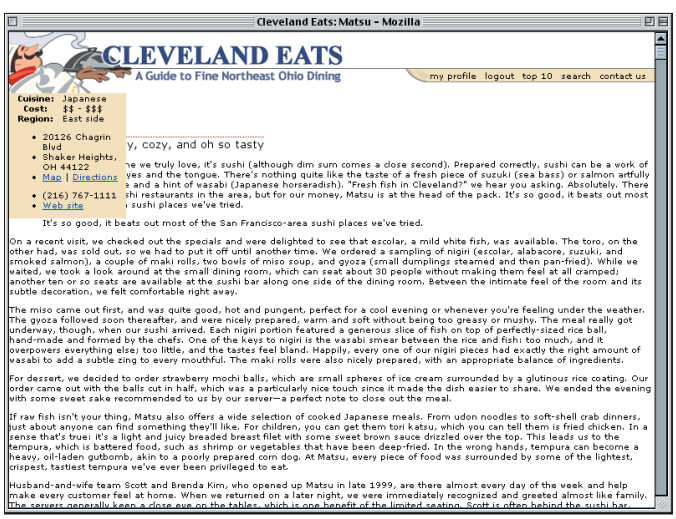


Abbildung 1.10

Die Seitenleiste ist platziert und in der Größe angepasst, aber es gibt noch genug zu tun.

Bemessen der Breite

Wir haben eine Breite in Pixel gewählt, weil beim Original-Design für die Seitenleiste die Breite in Pixel angegeben wurde, aber natürlich kann man das auch ganz anders bemessen. In positionsbasiertem Layout werden auch gerne Angaben in Prozent und em genommen.

Genau, der Inhalt überlappt! Das werden wir gleich reparieren, aber nun wollen wir die Seitenleiste erst ein wenig mehr füllen. Wir müssen einen Rahmen hinzufügen und auch die Trennung zwischen der Kante der Box und dem Inhalt darin neu erstellen.

```
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4; padding: 0.75em 0;
  border: 1px solid #600; border-width: 2px 1px 2px 0;}
```

Probleme beim Seiten-Padding

Wir haben das Padding für die Seite auf Null gesetzt, weil es sonst dem `width`-Wert hinzugerechnet wird, wie CSS es erfordert. Es ist leichter, Seitenränder oder Padding auf Elemente innerhalb der Seitenleiste einzustellen als direkt für das `div`-Element selbst.

Genauso werden wir auch den Rahmeneffekt neu erstellen, der im Tabellenlayout so viele Zellen benötigt hat. Was das Padding angeht, werden wir ems nehmen, damit das Padding bei Änderungen in der Textgröße skaliert wird und wir auch sicher sein können, dass wir die gleiche Art von vertikaler Ausrichtung bekommen, die im originalen Design zu sehen ist.

Die Listen müssen eindeutig mehr dem Original-Design angeglichen werden, in dem keine Gliederungspunkte (Bullets) waren. Wir können mit einer kleinen Regel die Bullets herausnehmen und die Einrückung angleichen.

```
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4; padding: 0.75em 0;
  border: 1px solid #600; border-width: 2px 1px 2px 0;}
#info ul {list-style: none; margin: 1em; padding: 0;}
#review h2 {color: #600; font-size: x-large;
  margin: 1em 0 0;}
```

Indem die Listen insgesamt ein em Seitenrand erhalten, werden sie voneinander getrennt und sind von den Rändern der Seitenleiste nach innen gerichtet, und genau das wollen wir erreichen. Jetzt brauchen wir nur noch darauf zu achten, dass die Tabelle oben in der Seitenleiste korrekt angelegt wird. Wir müssen die Label rechts ausrichten, und weil sie jetzt `th`-Elemente sind, geht das ganz leicht.

Eine Tabelle!

Die in der Tabelle enthaltene Information passt sehr gut zu einer solchen Struktur, also lassen wir alles wie gehabt. Wir könnten sie in eine Struktur ohne Tabelle konvertieren, die viele Floats einsetzt, aber das wäre wenig sinnvoll.

```
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4; padding: 0.75em 0;
  border: 1px solid #600; border-width: 2px 1px 2px 0;}
#info th {text-align: right;}
#info ul {list-style: none; margin: 1em; padding: 0;}
```

Ein wenig Padding bei den Tabellenzellen wird auch helfen, dass sich der Inhalt nicht so zusammenquetschen muss, also geben wir das auch hinzu.

```
#info th {text-align: right;}
#info td {padding: 0.125em;}
#info ul {list-style: none; margin: 1em; padding: 0;}
```

Nun bleibt nur noch eine Sache übrig: die Schrift für die Tabelle anpassen. Im Original-Design war sie ein wenig größer als alles andere und auch in einer anderen Schriftart. Wir werden die Tabelle einfach direkt stylen, wie in Abbildung 1.11 gezeigt.

```
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4; padding: 0.75em 0;
  border: 1px solid #600; border-width: 2px 1px 2px 0;}
#info table {font: small Arial, Verdana, Helvetica, sans-serif;}
#info th {text-align: right;}
```

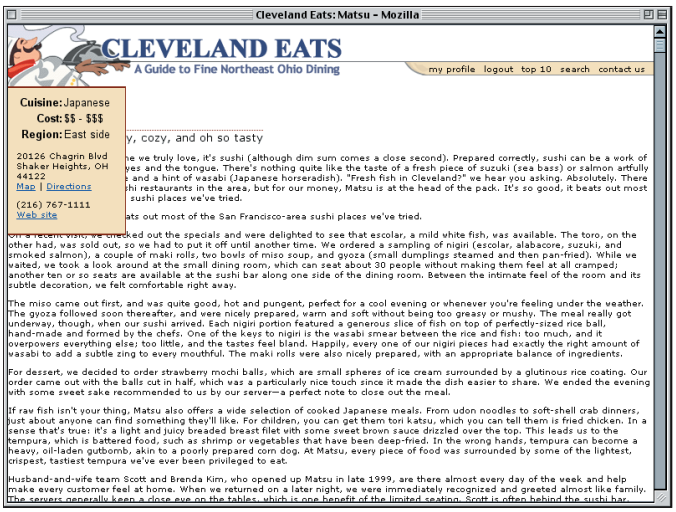


Abbildung 1.11

Nun sieht die Seitenleiste so gut wie früher aus, auch wenn sie den Text überlappt.

Wie bei der Navigationsleiste verleiht uns die Positionierung der Seitenleiste größere Flexibilität. Wir können sie ein wenig nach oben oder unten verschieben, indem wir den `top`-Wert ändern, seine Breite ausdehnen oder schrumpfen oder sie sogar auf die gegenüberliegende Seite schubsen, indem wir einfach die Eigenschaft `left` auf `right` ändern. Auf die gleiche Weise können wir die Breite und die Styles des Rahmens um die Seitenleiste im Handumdrehen ändern.

1.5.6 Restaurantkritik mit Style

Der Style der Seitenleiste mag nun abgeschlossen sein, aber sie sitzt immer noch ganz oben auf der Restaurantkritik, und das ist für dieses Projekt eindeutig keine gute Designentscheidung. Wir müssen den Text der Kritik ein wenig verschieben, aber bevor wir das tun, wollen wir uns noch mal unser Original-Design anschauen. Da hatten wir ein Spacer-GIF von 40 Pixel zwischen der Seitenleiste und dem Haupttext und einen 80-Pixel-Spacer auf der rechten Seite. Wir können beide Effekte in eine einzige Regel einbinden, indem wir das »Review«-`div` stylen.

```
#info ul {list-style: none; margin: 1em; padding: 0;}
#review {margin: 0 80px 2em 180px;}
#review h2 {color: #600; font-size: x-large;
  margin: 1em 0 0;}
```

Indem wir 40 Pixel der Breite der Seitenleiste von `140px` hinzufügen, bekommen wir `180px` für den linken Seitenrand, und natürlich hatte der rechte Rand klare `80px`. Dem unteren Rand von `2em` haben wir zugegeben, um unterhalb des Textendes ein wenig Extraraum zu schaffen.

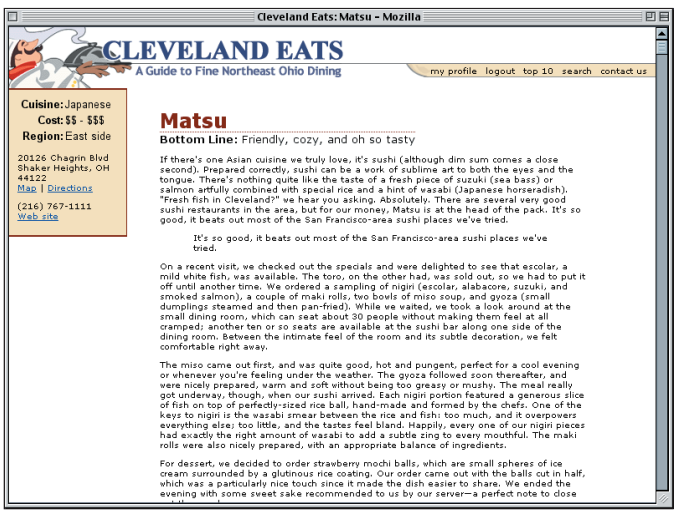
Nun, wo die Überlappungsprobleme gelöst sind, können wir uns auf den Text selbst konzentrieren. Wenn Sie sich die Abbildung 1.12 genauer anschauen, können Sie

sehen, dass die Zeile mit dem Untertitel nicht ganz an der »Region«-Zeile in der Seitenleiste ausgerichtet ist. Wegen dieser Ausrichtung brauchte das ursprüngliche Design solch ein kompliziertes Arrangement mit Zellen. Wir passen bloß den oberen Rand des Titels an. Über ein wenig Experimentieren kriegen wir heraus, dass die folgenden Werte einen ganz guten Effekt ergeben.

```
#review h2 {color: #600; font-size: x-large;
  margin: 1em 0 0; padding: 0 0 0.2em; line-height: 1em;}
```

ABBILDUNG 1.12

Die Spalten erscheinen, und der Text der Restaurantkritik ist lesbar.



Plattformsprünge

In Wirklichkeit ist der Text nicht völlig präzise. Die verwendeten Werte bringen in Windows-Browsern die gewünschten Ergebnisse, aber in Macintosh-Browsern ist die Ausrichtung ein wenig verschoben. Das scheint daran zu liegen, dass beide Betriebssysteme mit Schriften unterschiedlich umgehen, und es ist wahrscheinlich, dass andere Systeme ähnliche Abweichungen zeigen. Das illustriert exzellent einen wichtigen Punkt des Designs mit CSS: Man sollte sich auf Ausrichtungen dieser Art möglichst nicht verlassen.

Warum der Wert für die Zeilenhöhe? Zwar ist der nahe am Standard für alle Browser, aber es gibt keine Garantie dafür, dass es tatsächlich der Standard ist: Der eine Browser könnte 1,2 benutzen, ein anderer 1,25 und der dritte 1,175. CSS erfordert keinen spezifischen Wert als Standard, also können sich alle Browser da nach Belieben verhalten. Indem wir explizit einen Wert für `line-height` festlegen, umgehen wir alle potenziellen Probleme durch unterschiedliche Standards.

Jetzt wird es Zeit, dass wir uns die eigentlichen Textabsätze in der Restaurantkritik anschauen. Das Original-Design hat eine Reihe von Leerzeichen ohne Umbruch benutzt, um alle Absätze außer dem ersten einzurücken. Wir haben die alle entfernt, aber sie werden auch nicht gebraucht, denn `text-indent` erledigt die Arbeit für uns.

```
#review #summary {font-size: small; border-top: 1px dotted #600;
  display: inline;}
#review p {text-indent: 2em;}
</style>
```

Damit wird die erste Zeile aller `p`-Elemente im Restauranttext eingerückt, einschließlich des ersten und des Untertitels. (Sie erinnern sich, dass es in einem `p`-Element enthalten ist.) Glücklicherweise hatten wir dem ersten Absatz des Textes eine Klasse zugeordnet.

```
<p class="lead">
If there's one Asian food we truly love, it's sushi (although
dim sum comes a close second).
```

Wir können diese Klasse nutzen und dazu die Regel, die wir vorher schon für den Untertitel geschrieben haben, um das Einrücken der ersten Zeile in diesen beiden Elementen abzuschalten (siehe Abbildung 1.13).

```
#review #summary {font-size: small; border-top: 1px dotted #600;
  display: inline; text-indent: 0;}
#review p {text-indent: 2em;}
#review .lead {text-indent: 0;}
</style>
```

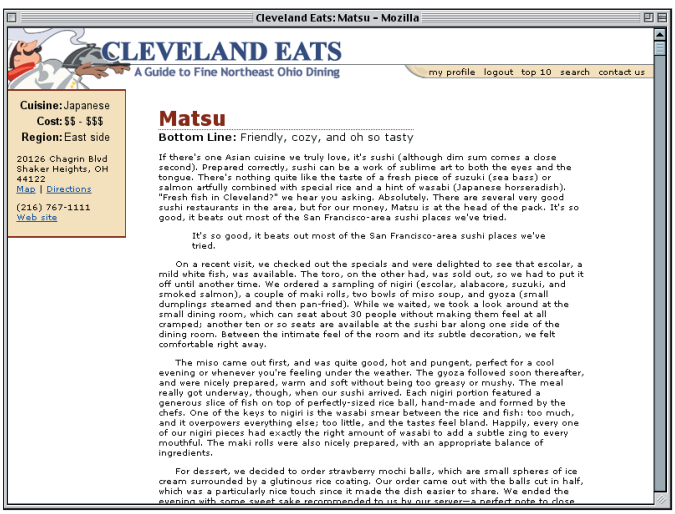


Abbildung 1.13

Einrücken der ersten Zeile bei den meisten, aber nicht allen Absätzen der Kritik.

1.5.7 Blickfänger und Designerweiterungen

An diesem Punkt sind wir fast fertig. Jetzt brauchen wir nur noch den Texteinschub so zu gestalten, dass er den Blick auch wirklich auf sich zieht. Momentan ist es bloß ein `blockquote`, der zwischen ein paar Absätzen hockt. Das lässt sich zum Glück einfach erledigen. Wir brauchen bloß den Float nach rechts auszurichten, Schriftgröße und -gewicht aufzupeppen und etwas Padding zuzugeben, damit der Text nicht zu dicht heranrückt.

```
#review .lead {text-indent: 0;}
blockquote.pull {float: right; width: 40%;
  padding: 1em 0 1em 5%; margin: 0;
  font-size: medium; font-weight: bold;}
</style>
```

Wenn Sie diese Styles mit dem originalen HTML-Design vergleichen, werden Ihnen einige Unterschiede auffallen, z.B. der, dass wir die Spacer.GIFs in Pixelgröße in ein

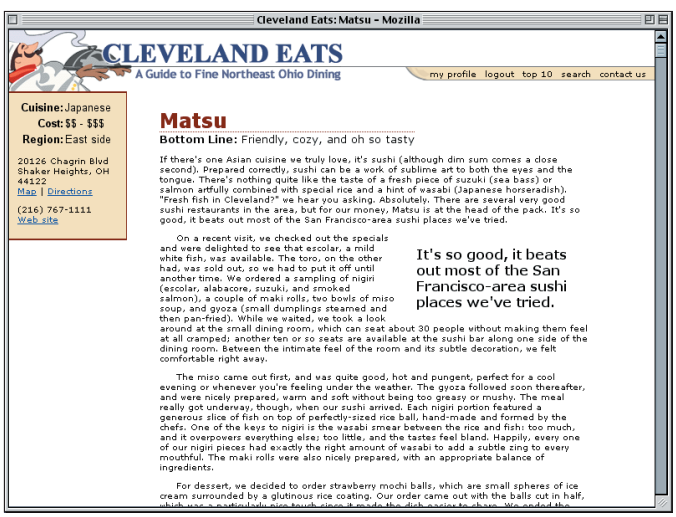
Padding mit `1em` konvertiert haben. Wir hätten stattdessen `10px` verwenden können, aber weil wir keine Pixel nehmen *müssen*, können wir doch auch etwas Flexibleres nehmen, oder?

Was das linke Padding von `5%` angeht, das hat scheinbar überhaupt nichts mit der ursprünglichen Breite der Spacer-Zelle von `15%` zu tun. Die originale Tabelle war ja `45%` breit, während die von uns gerade geschriebenen Styles eine Breite von `40%` erfordern.

Der Unterschied liegt in der Natur der Tabellen im Vergleich zu CSS-gestylten Elementen. Als wir diese Zelle mit der Breite `15%` eingerichtet haben, betrug die Breite 15 % der Tabelle, die wiederum 45 % von der Breite der Zelle hatte, in der sie saß. Die Prozentangaben beim Padding (wie auch die Seitenränder und `width` selbst) beziehen sich auf die Breite des jeweiligen Container-Elements, in diesem Fall das »Review«-`div`. Obendrein wird der Breite auch noch Padding hinzugefügt. Wenn wir also wollen, dass der Pullquote bis an 45 % der Breite des Container-Elements heranreicht, müssen wir das bei der Berechnung berücksichtigen. Also ergeben 40 % Breite plus 5 % Padding links 45 % Breite insgesamt. Das Ergebnis können Sie in Abbildung 1.14 sehen.

ABBILDUNG 1.14

Floats und Styles bilden den Pullquote des ursprünglichen Designs nach.



Nun, wo wir wieder beim Ausgangspunkt angekommen sind, wollen wir das Original-Design noch ein wenig verbessern. Durch die kleine Schriftgröße werden eine Menge Infos auf den Bildschirm gebracht, aber dadurch ist der Text auch schwerer lesbar. Unter HTML kann man da nicht viel machen, aber mit CSS können die Zeilen ganz einfach gestreckt werden. Wir brauchen bloß die `line-height` der Absätze ändern.

```
#review p {text-indent: 2em; line-height: 1.3;}
```

Das könnte sich jedoch möglicherweise auf das Absatz-Layout des Untertitels auswirken, also stellen wir es auf den Standard ein.

```
#review #summary {font-size: small; border-top: 1px dotted #600;
  display: inline; text-indent: 0; line-height: normal;}
```

Für einen kleinen Tick mehr Interaktivität wollen wir nun für die Links der Navigationsleiste einen Rollover-Style definieren. Als Ausgangspunkt legen wir fest, dass alle Links der Navigationsleiste, über denen der Mauszeiger ruht, als weißer Text auf einem dunkelblauen Hintergrund erscheinen (damit das zur Textfarbe im Seitenkopf passt), und erweitern den unteren Rahmen.

```
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 2px 1em 1px 0;}
#navbar a:hover {color: white; background: #336;
  border-bottom-width: 3px;}
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4; padding: 0.75em 0;
  border: 1px solid #600; border-width: 2px 1px 2px 0;}
```

Wenn wir das so machen, ergibt sich eine leichte Unausgeglichenheit in unseren grundlegenden Link-Styles der Navigationsleiste. Beachten Sie, dass im vorhergehenden Code das Padding bei den Links rechts `1em` beträgt und links `0`. Das heißt, dass der Rollover-Effekt bezogen auf den Text verschoben ist. Wir müssen das ausbalancieren, damit der Text in jedem Link zentriert ist, wie es in Abbildung 1.15 gezeigt wird.

```
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 2px 0.5em 1px;}
```

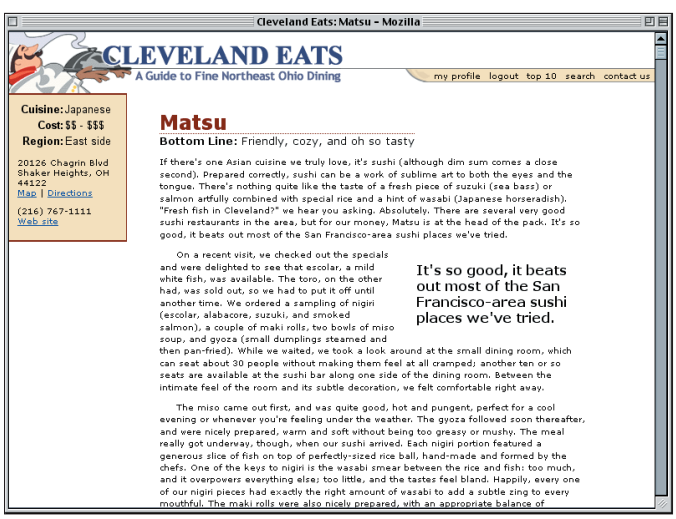


ABBILDUNG 1.15

Der Text wird auseinander gezogen und die Navigationslinks aufgepeppt.

Fehlende Effekte

Bugs im IE/Win verhindern oft, dass Elemente die Begrenzungen ihrer Eltern überschreiten, also wird der Rahmeneffekt mit 3 Pixeln wahrscheinlich nicht auftauchen. Die Technik haben wir trotzdem vorgeführt, weil sie ein gutes Beispiel für ein Design ist, das auch in weniger leistungsfähigen Browsern funktioniert, aber in fortschrittlichen Browsern einfach besser ausschaut.

1.5.8 Wieder beim Seitenkopf

Okay, wir haben also das Layout ganz neu gestrickt und das Original sogar ein wenig verbessert, aber was ist mit dem Masthead (Seitenkopf)? Er sitzt immer noch in einer Tabelle mit drei Zellen, und die Grafiken sind alle zerschnitten. Lassen Sie uns mal schauen, ob wir das verbessern können.

Wir können alle drei Abschnitte wieder zu einem Bild zusammensetzen, aber es wäre viel nützlicher, wenn wir den Koch und den Seitentitel in einem Bild verschmelzen und die Unterzeile (»A Guide to Fine Northeast Ohio Dining«) in ein anderes Bild legen. Wenn wir die Unterzeile extra haben, können wir sie durch Positionieren an eine beliebige Stelle schieben. Das führt also zu den Bildern in Abbildung 1.16.

ABBILDUNG 1.16

Durch das Zusammensetzen des Bildes und den separaten Untertitel verfügen wir über viel mehr Layoutmöglichkeiten.

Jetzt brauchen wir nur noch das HTML zu überarbeiten, damit die Bilder immer noch vorhanden sind, aber deutlich weniger Markup haben. Es sollte für den Moment ausreichen, beide in einen `div`-Bereich mit einigen passenden `id`-Attributen einzuschließen.

```
<body>
<div id="masthead">
<img src="masthead.gif" alt="Cleveland Eats">
<img src="subhead.gif" alt="A Guide to Fine Northeast Ohio
Dining" id="subhead">
</div>
<div id="navbar">
```

Im ersten Arbeitsgang platzieren wir die Unterzeile dort, wo sie hingehört. Um den genauen Ort festzustellen, zählen wir einfach die Pixel im Original-Design.

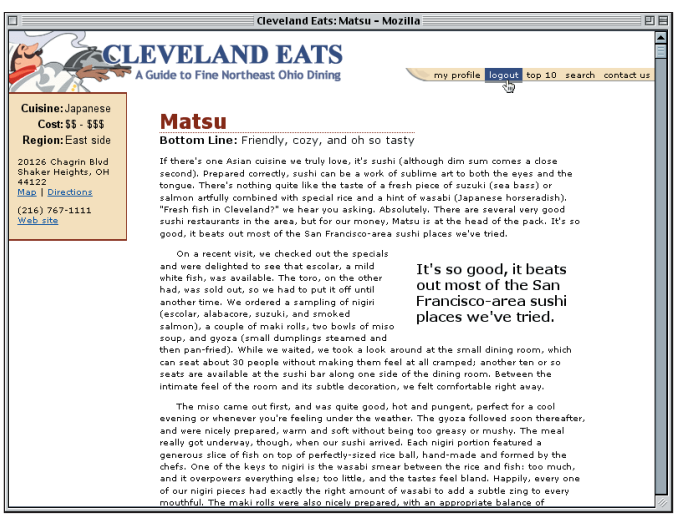
```
th, td {padding: 0;}
#subhead {position: absolute; top: 46px; left: 151px;}
#navbar {position: absolute; top: 44px; right: 0;
  padding: 2px 0 2px 32px; white-space: nowrap;
  background: #F0DFB4 url(tab-curve.gif) bottom left no-repeat;}
```

Dies platziert die obere linke Ecke der Unterzeilengrafik genau dort, wo wir sie haben wollen. Natürlich können wir sie auch ein wenig verschieben – sagen wir, ein oder

zwei Pixel nach links –, und das erreichen wir, indem wir mit den Werten von `top` und `left` herumspielen. Was das angeht, könnten wir sie wirklich ganz woanders hinschieben, vielleicht zu der oberen rechten Ecke der Seite oder direkt über die Navigationsleiste. Das werden wir aber jetzt nicht machen, damit können Sie später selbst einmal experimentieren.

Wo wir gerade dabei sind, werden wir auch die Bilder im Seitenkopf so einstellen, dass Blocklevel-Boxen entstehen. Wenn wir das tun, eliminieren wir die Wahrscheinlichkeit kleinerer Layout-Änderungen in Gecko-basierten Browsern (wie Mozilla), die das Bildlayout ein wenig strenger behandeln als die meisten anderen Browser. Wenn wir die Layout-Boxen als Blocklevel deklarieren, wird das die Unterschiede eliminieren, und wir bekommen das in Abbildung 1.17 gezeigte Layout.

```
th, td {padding: 0;}
#masthead img {display: block;}
#subhead {position: absolute; top: 46px; left: 151px;}
```



Absolut geblockt

Das absolut positionierte Bild in der Unterzeile generiert eigentlich schon eine Blocklevel-Box. Alle absolut positionierten Elemente machen das so – egal was für eine Art Box sie generiert hätten, wenn sie nicht positioniert gewesen wären.

ABBILDUNG 1.17

Die Unterzeile ist wie vorher platziert, doch wir können sie im Nu verschieben.

Eine Sache fehlt noch: diese verflixte Trennlinie. Wir wissen, dass sie 43 Pixel unter dem oberen Rand des Seitenkopfs sein soll, und das führt uns gewissermaßen in Versuchung. Wir könnten theoretisch eine explizite Höhe von `43px` für den Seitenkopf definieren, ihm einen unteren Rahmen geben und das `masthead.gif`-Bild über das `div`-Element laufen lassen. Das CSS dafür wäre dann:

```
#masthead {height: 42px; border-bottom: 1px solid #666;}
```

Wenn wir das so machen würden, dann müssten wir noch die Styles des »Review«-`div`s anpassen. Warum das? Weil der Seitenkopf nun 44 Pixel hoch wäre (die Höhe plus den unteren Rahmen) statt der 73 Pixel, die es jetzt hat. Also würde das `div` der »Review« um 29 Pixel nach oben verschoben. Das ist recht einfach zu reparieren: Oben ein Padding mit 29 Pixel steuert dem Effekt entgegen.

Leider stellt der Internet Explorer für Windows das Hindernis zur Verwendung dieser speziellen Technik dar, der die Höhe von `43px` ignoriert und das Seitenkopf-`div` ausdehnt, um das Bild einzuschließen. In Wirklichkeit behandelt der Explorer `height`, als wäre es `min-height`. Also bleiben uns zwei Optionen. Eine ist die Positionierung von `masthead.gif`, das dann aus dem normalen Flow herausgenommen wäre, und der Explorer würde dann bezogen auf die Höhe des Seitenkopf-`div`s das Richtige machen. Um das hinzukriegen, ist folgendes CSS nötig:

```
#masthead {height: 43px; border-bottom: 1px solid #666;}
#masthead img {position: absolute; top: 0; left: 0;}
#masthead #subhead {top: 46px; left: 151px;}
#review {margin: 0 80px 2em 180px; padding-top: 30px;}
```

Die andere Möglichkeit ist, die Dinge so zu lassen, wie sie in Abbildung 1.17 waren, und in den Hintergrund des Seitenkopfs nur ein Bild mit einem Pixel zu packen, das sich horizontal wiederholt:

```
#masthead {background: white url(stripe.gif) 0 43px repeat-x;}
```

Der Vorteil hierbei ist, dass über nur eine neue Regel die Trennlinie erscheint, und dafür ist jedes sich horizontal wiederholende Muster geeignet, aber der Nachteil ist, dass dafür ein weiteres Bild geladen werden muss, damit der Effekt funktioniert. Was Sie bevorzugen, können Sie selbst entscheiden; auf jeden Fall sollten Sie das in Abbildung 1.18 gezeigte Ergebnis bekommen.

ABBILDUNG 1.18

Der Seitenkopf wird ohne Tabellen völlig neu aufgebaut.

Weil Pearson bereits eine Menge Geld für gutes Papier und Farbdruck springen lässt, werden wir hier das Listing für die Version des fertigen Stylesheets mit dem Hintergrundbild aufführen – sie ist kürzer als die andere.

Listing 1.2 Der Ansatz mit dem Hintergrundbild

```
body {font: x-small Verdana, Arial, Helvetica, sans-serif;
  margin: 0; padding: 0;}
table {width: 100%;}
th, td {padding: 0;}
#masthead {background: white url(stripe.gif) 0 43px repeat-x;}
#masthead img {display: block;}
#subhead {position: absolute; top: 46px; left: 151px;}
#navbar {position: absolute; top: 44px; right: 0;
  padding: 2px 0 2px 32px; white-space: nowrap;
  background: #F0DFB4 url(tab-curve.gif) bottom left no-repeat;}
#navbar b {display: none;}
#navbar a {text-decoration: none; color: #000;
  border-bottom: 1px solid gray;
  padding: 2px 0.5em 1px;}
#navbar a:hover {color: white; background: #336;
  border-bottom-width: 3px;}
#info {position: absolute; top: 73px; left: 0; width: 140px;
  background: #F0DFB4; padding: 0.75em 0;
  border: 1px solid #600; border-width: 2px 1px 2px 0;}
#info table {font: small Arial, Verdana, Helvetica, sans-serif;}
#info th {text-align: right;}
#info td {padding: 0.125em;}
#info ul {list-style: none; margin: 1em; padding: 0;}
#review {margin: 0 80px 2em 180px;}
#review h2 {color: #600; font-size: x-large;
margin: 1em 0 0; padding: 0 0 0.2em; line-height: 1em;}
#review #summary {font-size: small; border-top: 1px dotted #600;
  display: inline; text-indent: 0; line-height: normal;}
#review p {text-indent: 2em; line-height: 1.3;}
#review .lead {text-indent: 0;}
blockquote.pull {float: right; width: 40%;
  padding: 1em 0 1em 5%; margin: 0;
  font-size: medium; font-weight: bold;}
```

Wo liegt der Unterschied?

Mit welcher Technik ist nun also die Abbildung 1.18 geschaffen worden? Das sollte eigentlich egal sein, aber wenn Sie es genau wissen wollen, gucken Sie in der Datei `ch0118.html` nach.

1.6 Bewertung der Vorteile

Sie begegnen einem Konvertierungsprozess wie dem, den wir in diesem Kapitel beschrieben haben, vielleicht mit einer gewissen Irritation. »Warum die Mühe, wenn's doch auch mit Tabellen klappt?« ist die übliche Frage. Dafür gibt es zwei gute Gründe.

Erstens ist die Dokumentstruktur deutlich sauberer und darum ist die Bearbeitung und Wartung viel einfacher. Nehmen wir an, dass Sie irgendwo in Ihrem Markup ein nicht geschlossenes Element haben, das das gesamte Layout durcheinander bringt. Möchten Sie sich lieber durch ein HTML-Design graben, das mit `font`-Tags und ineinander verschachtelten Tabellen vollgestopft ist, oder durch die relativ saubere Struktur, die wir am Ende des Konvertierungsprozesses vorfinden? Wahrscheinlich letzteres.

Trefferbereich

Manche der Werte für »Server-Hits« sind als Spanne angegeben, und das liegt an mehrfach verwendeten Bildern. Im reinen HTML-Design gab es beispielsweise 18 `spacer.gif`-Bilder. Die meisten Browser laden das nur einmal herunter und nutzen ab da eine gecachte Version des Bildes, aber ein Browser mit abgeschaltetem oder ganz fehlendem Cache muss das Bild jedes Mal laden, wenn es im HTML referenziert wird.

Zweitens gibt es merkliche Einsparungen im Hinblick auf die Dateigröße. Tabelle 1.1 vergleicht die Dateigröße, Anzahl der Elemente und Server-Hits, die für jede der drei Herangehensweisen nötig sind (oder sein können): das reine HTML-Design, das konvertierte Dokument mit eingebetteten Stylesheets und das konvertierte Dokument, bei dem das Stylesheet extern angelegt und verlinkt wurde.

Tabelle 1.1 Qualitativer Vergleich der drei Ansätze

Methode	Größe*	Zeichen	Server-Hits	Bilder	Tabellen	Font-Tags
reines HTML	100 %	8,994	6–23	22	7	20
				18 Wiederholungen		0
HTML + CSS	75,4 %	6,785	5	4	1	0
HTML + verlinktes CSS	59,8 %	5,375 + 1,435	5–6	4	1	0

* Verglichen mit der reinen HTML-Methode; bezieht sich nur auf die Größe des HTML-Dokuments

Sogar wenn das Stylesheet im Dokument verbleibt, gibt es bei der Dateigröße eine Reduktion um fast 25 %. Wenn das Stylesheet in eine externe Datei verschoben wird, sind die Vorteile sogar noch überzeugender, denn dann hat sie nur noch 60 % der Originalgröße. Egal wie, auf jeden Fall wird die Seite schneller heruntergeladen, die Benutzer sind zufriedener, und die Bandbreite wird reduziert, was weniger kostet.

Der Wert 59,8 % benötigt eine kurze Erklärung. Es wird nur das HTML einbezogen, nicht das externe Stylesheet. Das liegt daran, dass Browser normalerweise externe Stylesheets cachen, nachdem sie das erste Mal geladen wurden. Darum wird der Server nicht nach dem externen Stylesheet gefragt, nachdem der Benutzer die Seite (die dieses Stylesheet nutzt) das erste Mal vom Server geladen hat. Für dieses erste Herunterladen kommen die Einsparungen bei der Bandbreite der HTML- und CSS-Version recht nahe, die immer noch eine Reduzierung um ein Viertel beträgt.

Zusätzlich zu diesen Einsparungen ist die positionierte Version des Layouts viel, viel leichter zu ändern, wenn wir das wünschen. Ob wir die Breite der Seitenleiste anpassen, die Unterzeile verschieben oder gar das allgemeine Layout überarbeiten wollen – alles läßt sich ausgehend vom Basislayout leicht und schnell ändern. Alles in allem erscheint es die Mühe wert zu sein, das unternommen zu haben.

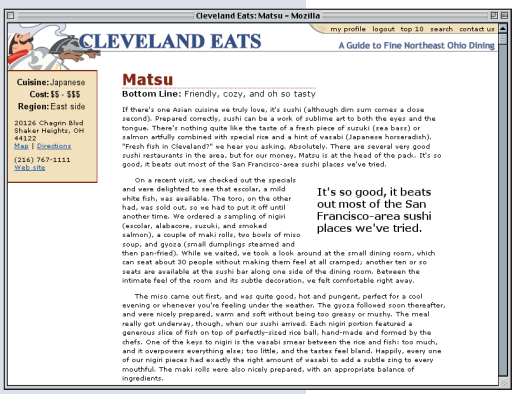
1.6.1 Eine Warnung

Zu diesem Layout muss ein Wort der Warnung gesagt werden. Falls die Spalte mit dem Text der Restaurantkritik kürzer als die Seitenleiste sein sollte, können merkwürdige Dinge passieren. In Tabellen weisen alle Zellen einer Reihe die gleiche Höhe auf; wenn wir also die Seitenleiste in eine Zelle und den Text der Kritik in eine andere legen, hätten beide Zellen die Höhe der größeren Zelle.

Bei der Positionierung gibt es keine Möglichkeit, zwei Elemente so zu verbinden. Wenn Sie erst einmal ein Element aus dem Dokumentfluss genommen haben, wird seine Höhe durch seinen Inhalt und seine Styles bestimmt – Punktum! Es wird nicht durch andere Elemente beeinflusst noch beeinflusst es andere Elemente. Also wäre die Positionierung eine schlechte Wahl für ein Layout, in dem die beiden Spalten die gleiche visuelle Höhe haben müssen oder wo es Inhalt gibt, der nach beiden Spalten folgt, und Sie wissen nicht, welche Spalte im endgültigen Layout größer wird. Das heißt nicht, dass Positionierung nutzlos ist. Mehr und mehr hochwertige Sites wie *Wired News*, *Sprint PCS* und *Quark* verwenden für das Layout CSS-P. Sie müssen einfach nur wissen, wo Positionierung sich lohnt und wo nicht, damit Sie die beste Entscheidung für Ihre Layouts treffen können.

1.7 Spielwiese

Nehmen Sie sich noch einmal die Styles vor, die wir in diesem Projekt geschaffen haben, und versuchen Sie, die folgenden Modifikationen zu erreichen, ohne die Struktur des Dokuments zu ändern. Beachten Sie, wie leicht das auf diese Weise geht, anstatt die Tabellen neu schreiben zu müssen.



1. Bewegen Sie die Navigationsleiste in die obere rechte Ecke der Seite und stellen Sie die Unterzeile darunter, so dass die Grundlinien der Unterzeile und der Haupttitel in einer Flucht sind. Weil das etwas Raum unter dem Seitenkopf frei räumt, bewegen Sie den Text der Kritik nach oben, damit Sie nicht zu viel Bildschirmfläche verschwenden.

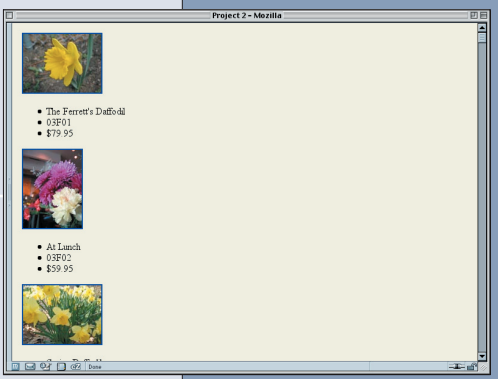
2. Spiegeln Sie das Layout, so dass die Seitenleiste rechts und der Text der Kritik links ist. Denken Sie daran, dass der Koch genau da abgeschnitten wird, wo jetzt die Seitenleiste sitzt, also wird noch eine visuelle Nacharbeitung nötig sein. Bonuspunkte gibt es, wenn Sie den oberen Rahmen des Untertitels so ausdehnen, dass er die Info-Box berührt, aber der Text aus der Restaurantkritik der Seitenleiste dabei nicht näher als 40 Pixel kommt.

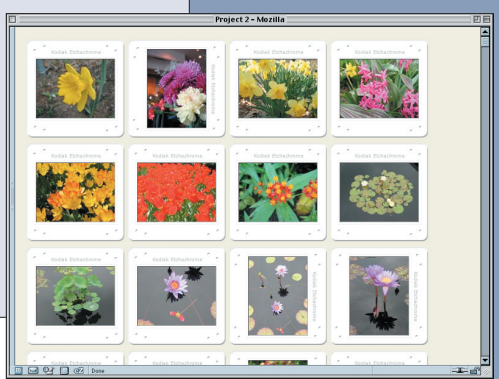
2

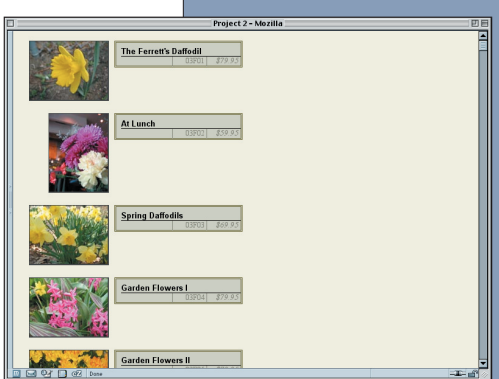
Eine Bildersammlung wird gestylt

Alle Fotografien existieren, um uns daran zu erinnern, was wir vergessen. Darin – wie auch bei anderen Aspekten – sind sie das Gegenteil der Malerei. Gemälde halten fest, was der Maler erinnert. Weil jeder von uns andere Sachen vergisst, kann ein Foto mehr als ein Gemälde seine Bedeutung verändern, je nach dem, wer es anschaut.

- John Berger







Obwohl nicht jeder die eigenen Fotografien ins Netz stellt, sind solche Sammlungen eine interessante Herausforderung für ein Layout. Jedes Foto und die damit verbundenen Informationen formen eine kleine, abgeschlossene Einheit, die nichtsdestotrotz mit Rücksicht auf die anderen Fotos auf der Seite entworfen werden muss. Auf gewisse Weise sind sie wie Portale, außer dass jeder »Kasten« in diesem Portal statt zu den aktuellen Schlagzeilen oder Sportergebnissen zu mehr Informationen über ein Foto führt. Fotosammlungen erinnern ebenfalls an eine andere, deutlich anspruchsvollere Layout-Herausforderung: die eines Warenkataloges für die Produkte einer E-Commerce-Site. Manchmal sind die Fotos sogar selbst zum Verkauf stehende Produkte, und von dieser Annahme gehen wir bei unserem Projekt aus.

2.1 Projektziele

In diesem Projekt schauen wir uns an, wie wir eine Sammlung von Fotografien zum Verkauf präsentieren können. Unser Kunde hat uns die folgenden Bedingungen gestellt:

- Es müssen drei unterschiedliche Präsentationsformen möglich sein: eine Kontaktbogen- oder Thumbnailansicht für den Künstler, damit er prüfen kann, was alles verfügbar ist und damit bei seinen Kollegen angeben kann, eine Galerieansicht für Benutzer, damit sie eine Übersicht der Angebote bekommen, und eine detaillierte Katalogansicht für die Bestellung.
- In der Galerie- und der Kontaktbogenansicht sollten so viele Fotos wie möglich »über der Falz« erscheinen und möglichst ohne horizontales Scrollen sichtbar sein, egal wie groß das Browserfenster ist. Es ist akzeptabel, nur das Foto und seinen Namen in dieser Ansicht zu sehen. Allerdings sollten die Bilder sich selbst in einem regelmäßigen Raster anordnen.
- In der Katalogansicht sollte jedes Foto mit Namen, Katalognummer und Preis angezeigt werden. In dieser Ansicht ist Scrollen erträglich.
- Allen drei Ansichten sollte das gleiche Markup zugrunde liegen, weil unser Kunde kein Geld für eine dynamische Site ausgeben möchte und darum das Seiten-Markup nur einmal produziert haben will.

Für dieses Projekt arbeiten wir nur mit dem Layoutbereich der Fotosammlung, also brauchen wir uns nur um diesen Projektteil kümmern. Wir werden davon ausgehen, dass das Layout sich in einer zentralen Hauptspalte in einem größeren Layout befinden wird, aber das ändert nicht wirklich etwas für dieses Projekt.

Aufgrund der Einschränkungen des Projekts, insbesondere im Bereich der Galerie- und der Kontaktbogenansicht, werden wir keine Tabellen nutzen können, um diese Fotos anzuordnen. Warum nicht? Wegen der Anforderung, so viele Bilder wie möglich »über der Falz« (das heißt, wenn die Seite im Fenster des Browsers geladen wird) unterzubringen.

So werden wir die Bilder und ihre Informationen für diese beiden »Kompakt«-Ansichten mit Floats statt mit Tabellen einrichten. Durch Floats können wir so viele Bilder in jeder »Reihe« unterbringen, wie in das Browser-Fenster passen. Anders gesagt wird ein Anwender mit einem Browserfenster mit 800 x 600 etwa vier Bilder pro Reihe angezeigt bekommen, während in ein Fenster mit 1280 x 1024 sechs oder sieben passen. Dieses »fließende« Verhalten können wir mit Tabellen nicht erreichen, wohl aber mit Floats. Obendrein können wir die Floats so einrichten, dass alle die gleiche Breite haben. Damit können wir erreichen, dass sie sich selbst rasterähnlich anordnen.

2.2 Vorbereitungen

Laden Sie die Dateien für Projekt 2 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch02proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und erneut laden.

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

2.3 Die Grundlagen

Zuallererst sollten wir einen Blick auf das Markup werfen, mit dem wir arbeiten werden. Hier sind die ersten beiden Bilder und Informationen im Dokument:

```
<div class="pic ls"><a href="orig/img01.jpg" class="tn"><img
  src="tn/thumb01.jpg" alt=""></a><ul>
<li class="title">The Ferrett's Daffodil</li>
<li class="catno">03F01</li>
<li class="price">$79.95</li>
</ul></div>
<div class="pic pt"><a href="orig/img02.jpg" class="tn"><img
  src="tn/thumb02.jpg" alt=""></a><ul>
<li class="title">At Lunch</li>
<li class="catno">03F02</li>
<li class="price">$59.95</li>
</ul></div>
```

Whitespace Blues

Schauen Sie sich das Markup genau an: Achten Sie darauf, dass das Element `ul` genau gegen die Verknüpfung davor geschoben wurde, und das abschließende Tag des `div` ist dahinter. Das wurde gemacht, um gewisse Bugs in älteren Explorer-Versionen zu vermeiden, die durch Whitespace ausgelöst werden. Es ist bedauerlich, aber Whitespace betrifft immer noch einige ältere Browser, und manchmal kann ein Hinzufügen oder Wegnehmen von Whitespace mysteriöse Layout-Probleme lösen.

Das sind eine Menge Klassen, und wir müssen nun herausfinden, was sie alle bedeuten. Zum Glück haben wir einen Style-Guide.

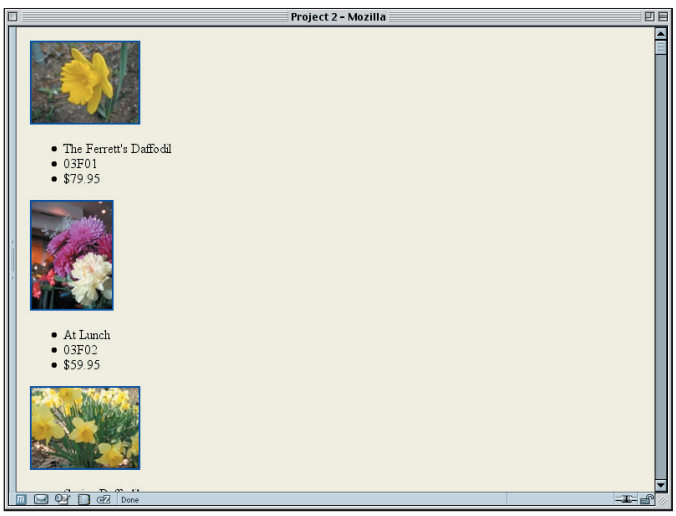
- `pic` kennzeichnet alle `div`s, die ein Bild und dazu gehörige Informationen enthalten. Das hilft uns, diese `div`s von allen anderen getrennt zu halten, die vielleicht noch verwendet werden.
- `ls` heißt, das Bild ist im Querformat (ls = *landscape*, Landschaft), also breiter als hoch, und `pt` bezieht sich auf das Hochformat (pt = *portrait*, Porträt) – höher als breit.
- `tn` markiert eine Verlinkung, die den das Thumbnail-Bild umgebenden Hyperlink darstellt.
- `title` markiert den Namen des Bildes, `catno` seine Katalognummer und `price` ... okay, Sie verstehen schon.

Die wirklich wichtigen Klassen sind `ls` und `pt`, wie wir bald sehen werden, aber alle werden sich als ganz praktisch erweisen. Wir werden diese Klassen beispielsweise nutzen, um Breite und Höhe der Bilder einzustellen. Wie Sie sehen, wird das im HTML nicht ausgedrückt. Wir wissen, dass unsere querformatigen Thumbnails 128 Pixel breit und 96 Pixel hoch sind (das Hochformat hat 96 x 128), aber das müssen wir im CSS auch so festlegen, bevor das Projekt fertig ist.

Zu Anfang wollen wir ein paar grundlegende Styles für Body und Footer festlegen. Für das Element `body` werden wir einfach einen leicht getönten Hintergrund und Ränder hinzufügen. Wir wissen bereits, dass wir häufig `float` verwenden werden, und wir wollen, dass der Footer nach den Bildern erscheint, also werden wir dort `clear` einsetzen. Die Auswirkungen sehen Sie in Abbildung 2.1.

```
<style type="text/css">
body {background: #EED; margin: 1em;}
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
</style>
```

ABBILDUNG 2.1
Die ersten Schritte



Diese Styles werden sich bis zum Ende des Projekts nicht ändern, also brauchen wir uns darum nicht mehr zu kümmern. Das haben wir nun erledigt, also können wir zur Sache kommen!

2.4 Die Kontaktbogenansicht wird erstellt

In dieser Phase können wir die Dokumentstruktur selbst sehen: Bilder, gefolgt von unsortierten Listen. Wir wollen uns zu diesem Zeitpunkt der Erstellung eines »Kontaktbogen«-Layouts zuwenden, in dem alle Bilder rasterförmig angeordnet sind. Dann können wir auf einen Blick so viele Bilder wie möglich sehen.

2.4.1 Voll im Fluss

Da wir keine Tabelle haben wollen, ist die nahe liegende Lösung, die Bilder mit Floats einzubauen. Wir wissen, dass die Bilder nicht mehr als 128 Pixel breit oder hoch sind, also legen wir unsere `div`s auf 128 x 128 aus und geben ihnen weißen Hintergrund

Aus dem Rahmen gefallen

Nicht alle Browser legen einen Rahmen um ein verlinktes Bild, aber einige doch, also ist es eine gute Idee, den Rahmen explizit loszuwerden. Bei Browsern, die sowieso keinen Rahmen machen, ist es dann auch egal.

und schwarze Rahmen. Was die Rahmen angeht – auch die blauen Rahmen der Grafikreferenzen werden wir uns vom Hals schaffen.

```
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic {float: left; height: 128px; width: 128px;
  background: white;
  border: 1px solid black;}
div.pic img {border: none;}
</style>
```

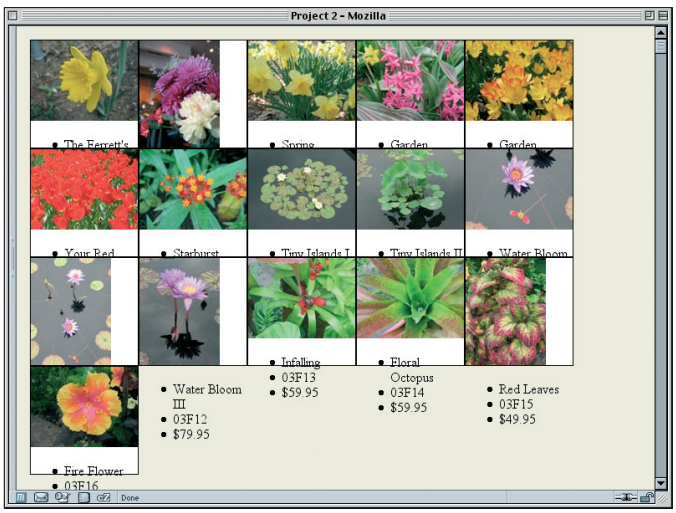


ABBILDUNG 2.2

Die `divs` der Bilder mit Floats

Nun haben wir für unseren Kontaktbogen bereits einen Riesenschritt geschafft, aber es gibt ein offensichtliches Problem – die Listen! Weil wir die `divs` zu einer bestimmten Höhe gezwungen haben, gibt es nicht genug Platz für die Listen, also laufen sie aus den `divs` heraus und bringen uns das ganze Layout durcheinander. Wir müssen sie also fürs Erste entfernen, also werfen wir sie einfach aus der Darstellung hinaus.

```
div.pic img {border: none;}
div.pic ul {display: none;}
</style>
```

Damit wird die Anzeige dieser Information generell unterbunden. Um die Listen werden wir uns in späteren Änderungen an den Styles kümmern, aber nun sind wir sie erst einmal los.

2.4.2 Zwischenräume und Zentrierungen

Die Fotos bilden dank der Floats ein Raster, aber alles wirkt ein wenig zu beengt. Wir verteilen die Bilder ein wenig, indem wir den Floats Ränder zuweisen.

Ein Blick auf den Explorer

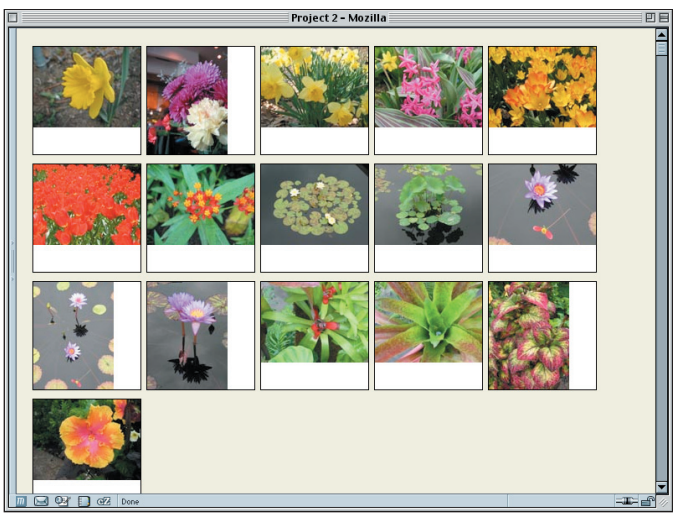
Im Explorer für Windows werden die `divs` von den Listen ausgedehnt, anstatt aus ihnen herauszulaufen, was zu einem ganz anderen, aber ebenso schlechten Layout führt. Das ist ein Bug im Explorer 5+, der `height` so behandelt, als wäre es `min-height` (was der Explorer ironischerweise nicht unterstützt).

```
div.pic {float: left; height: 128px; width: 128px;
  margin: 5px 3px; background: white;
  border: 1px solid black;}
```

Weil die Floats für Ränder sich nicht überschneiden, wird der tatsächliche Raum zwischen zwei Floats, die direkt nebeneinander stehen, 6 Pixel betragen (`3px + 3px`), und es werden 10 Pixel zwischen einem Float und dem nächsten darüber oder darunter sein. Dies können Sie natürlich so anpassen, wie Sie den Zwischenraum gerne hätten. Was wir bisher geschafft haben, können Sie in Abbildung 2.3 sehen.

ABBILDUNG 2.3

Die Thumbnails werden auseinandergezogen.



Es wird schon immer besser, aber die Bilder sehen jetzt immer noch komisch aus, entweder ganz oben oder links am Rand des Kastens. Es sähe deutlich besser aus, wenn sie in jedem Kasten zentriert wären.

Dafür sollten wir aber erst die Größe der Bilder definieren. Das können wir mit zwei einfachen (und sehr ähnlichen) Regeln erledigen.

```
div.pic img {border: none;}
div.ls img {height: 96px; width: 128px;}
div.pt img {height: 128px; width: 96px;}
div.pic ul {display: none;}
```

Wir haben hier bloß klar gestellt, was wir bereits wissen, aber der Browser nicht: dass die querformatigen (`ls`) Bilder 96 Pixel hoch und 128 Pixel breit sind und dass es bei den hochformatigen Bildern (`pt`) genau anders herum ist.

Erinnern Sie sich noch daran, dass wir unsere `div`s mit 128 x 128 Pixel definiert haben? Jetzt brauchen wir zum Zentrieren den Bildern bloß noch ein paar Ränder zuzuweisen. Die Differenz zwischen 128 und 96 beträgt 32, und die Hälfte davon ist 16. Darum brauchen die Querformatbilder `16px` am oberen und unteren Rand und

die Hochformatigen `16px` für linken und rechten Rand. Das Ergebnis können Sie in Abbildung 2.4 sehen.

```
div.ls img {height: 96px; width: 128px; margin: 16px 0;}
div.pt img {height: 128px; width: 96px; margin: 0 16px;}
```

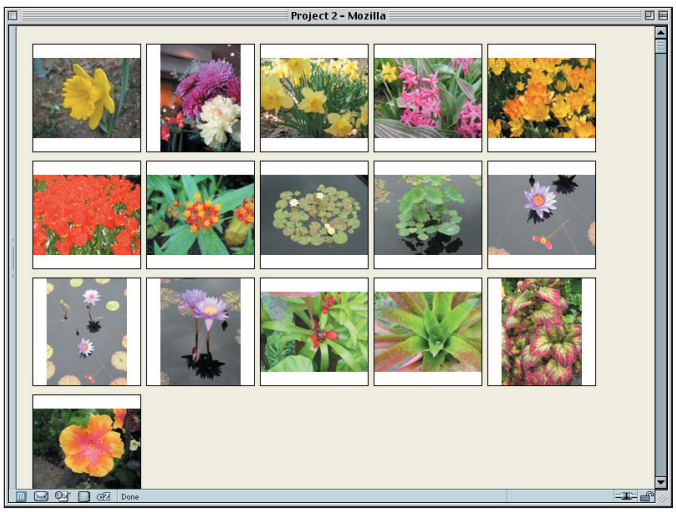


Abbildung 2.4
Zentrieren der Thumbnails

2.4.3 Dias in Style

Das ist schon ein ganz hübsches Layout, aber es kann noch besser werden. Bei den `divs` werden wir nun noch extra Padding zufügen, damit um die Thumbnails nur Weiß ist. Wir halten uns an unser Konzept mit den Zweierpotenzen und fügen 16 Pixel Padding hinzu.

```
div.pic {float: left; height: 128px; width: 128px;
  padding: 16px; margin: 5px 3px; background: white;
  border: 1px solid black;}
```

An diesem Punkt wirkt der Kontaktbogen wie eine Sammlung von 35mm-Dias, also greifen wir diese Idee auf. Zuerst geben wir den Bildern wieder einen Rahmen, mit dem sie dann ausschauen, als ob sie in einem Diarähmchen steckten.

```
div.pic img {border: 1px solid;
  border-color: #444 #AAA #AAA #444;}
```

Mit dieser Änderung kriegen alle Bilder einen dunkelgrauen oberen und linken Rahmen und einen hellgrauen rechten und unteren Rahmen. Dieser 3D-Effekt ist schon ganz nett. Allerdings kommt durch diese Regel ein kleines Ungleichgewicht ins Spiel. Die Bilder plus Rahmen sind nun länger als 128 x 96 Pixel und umgekehrt; jetzt sind sie 130 x 98 Pixel, weil die Rahmen an die Höhe und Breite der Bilder angesetzt werden. Um dem zu begegnen, müssen wir die Deklarationen für Höhe und Breite sowohl für die `divs` als auch für das Padding ändern.

Ein anderer Ansatz

Statt der gezeigten Technik hätten wir auch eine einzige Farbe und den Border-Style `inset` für den Einfügerahmen nehmen können, aber dieser Ansatz hat einen Nachteil: Browser dürfen die Farben für `inset` (wie auch für `outset`, `groove` und `ridge`) nach Belieben modifizieren. Wie vorherzusehen war, machen sie das alle unterschiedlich. Darum ist es in Situationen, bei denen es auf die farbliche Abstufung eines Rahmens ankommt, besser, den Rahmen `solid` zu machen und die Farben so einzustellen, wie Sie es wünschen.

```
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px; background: white;
  border: 1px solid black;}
```

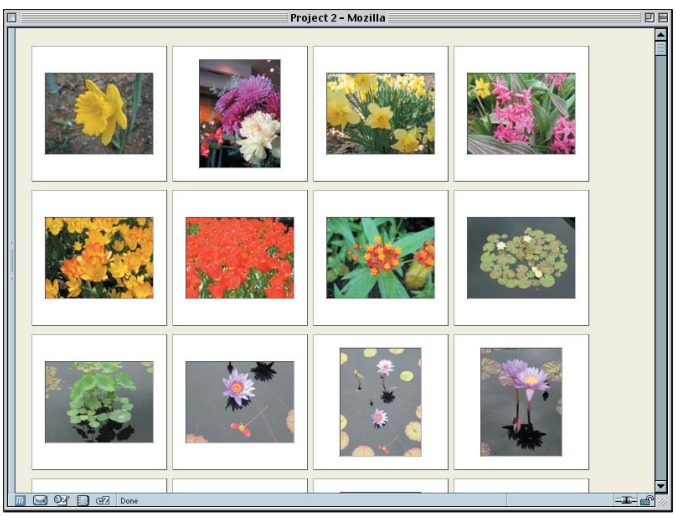
Durch diese kleinen Änderungen haben wir das frühere Gleichgewicht wieder hergestellt. Um den Effekt abzuschließen, wollen wir für die Rahmen der `div`s einen Outset-Effekt schaffen. Wir nehmen hier einfach die gleichen Farben, wie wir sie beim Inset-Effekt gehabt haben – bloß umgekehrt.

```
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px; background: white;
  border: 1px solid; border-color: #AAA #444 #444 #AAA;}
```

Bitte beachten Sie, dass wir das Schlüsselwort `black` aus der `border`-Deklaration entfernt haben. Dank der Deklaration für `border-color` brauchen wir es nicht mehr; um die Datei klein zu halten, haben wir es also herausgenommen. Das Ergebnis sehen wir in Abbildung 2.5.

ABBILDUNG 2.5

Eine Seite mit Dias – so wirkt es jedenfalls!



Das sieht 35mm-Dias schon ziemlich ähnlich, aber wir können den Effekt noch mehr steigern. Statt dass wir uns nur auf Hintergrundfarben und Rahmen verlassen, können wir diese entfernen und dafür ein Hintergrundbild nehmen.

Wie das? Wir kennen bereits die Ausmaße der `div`s, wenn erst einmal alles erledigt ist: Sie betragen 162 x 162 Pixel. Querformatige Dias haben beispielsweise die folgenden Maße entlang der horizontalen Achse:

128 px `img` Breite + 2 px `img` Rahmen + 30 px `div` Padding + 2 px `div` Rahmen = 162 Pixel insgesamt.

Da wir die Rahmen der `div`s entfernen wollen, können die auch jetzt gleich rausfliegen, also bleiben uns noch 160 x 160 Pixel. Darum werden wir die Hintergrundbilder

für die `divs` auch in dieser Größe anlegen. Weil wir zwei verschiedene Arten von Bildern haben (Hoch- und Querformat), brauchen wir zwei verschiedene Hintergründe. Die werden wir `frame-pt.gif` und `frame-ls.gif` nennen. Dabei bleibt es gleich, ob diese durch Einscannen echter Diarahmen oder durch das Erstellen einer Vorlage in Photoshop geschaffen werden. Wir brauchen nur etwas, das wie ein Rahmen aussieht.

Sie finden die Bilder namens `frame-pt.gif` und `frame-ls.gif` in den Projektdateien auf der Website für dieses Buch.

Nachdem wir die benötigten Hintergrundbilder erstellt haben, brauchen wir sie nur noch den Styles hinzufügen. Wir fangen damit an, dass wir das gleiche Bild auf alle `divs` anwenden und auch die Styles für unsere Rahmen entfernen.

```
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px;
  background: url(frame-ls.gif) center no-repeat;}
```

Das ist ganz prima für die querformatigen Thumbnails, aber es sieht fürchterlich aus, wenn es auf die hochformatigen Bilder angewandt wird. Also werden wir einfach das Bild für die hochformatigen Thumbnails durch eine kleine Regel ersetzen.

```
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px;
  background: url(frame-ls.gif) center no-repeat;}
div.pt {background-image: url(frame-pt.gif);}
div.pic img {border: 1px solid; border-color: #444 #AAA #AAA #444;}
```

Dies wird den Wert für das Hintergrundbild überschreiben, während die anderen Schlüsselwörter (`center no-repeat`) unverändert bleiben, was die Deklaration `background-image: url(frame-pt.gif)` funktional äquivalent zur Deklaration `background: url(frame-pt.gif) center no-repeat` macht. Die Abbildung 2.6 zeigt das Ergebnis.

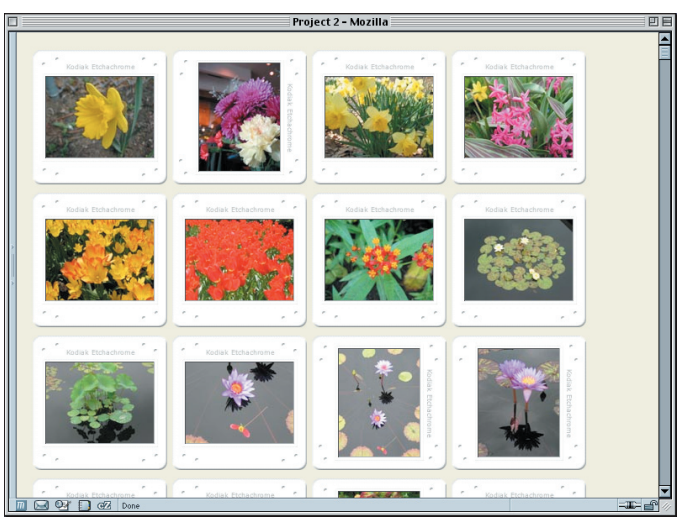


ABBILDUNG 2.6

Jetzt sehen sie wirklich schon wie Dias aus!

Jetzt wäre ein guter Zeitpunkt, Ihre Arbeit in einer separaten Datei abzuspeichern, weil wir im nächsten Abschnitt viele dieser Styles entfernen und auch einige neue hinzufügen werden.

Das ist ein ganz schicker Effekt, nicht wahr? Ganz klasse ist auch, dass Sie den »Diarahmen«-Hintergrund später durch ein besseres Bild ersetzen können, indem Sie einfach die Bilddateien aktualisieren. Das Listing 2.1 zeigt das vollständige Stylesheet für die von uns geschaffene Diasammlung.

Listing 2.1 Das vollständige »Dia«-Stylesheet

```
body {background: #EED; margin: 1em;}
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px;}
  background: url(frame-ls.gif) center no-repeat;}
div.pt {background-image: url(frame-pt.gif);}
div.pic img {border: 1px solid; border-color: #444 #AAA #AAA #444;}
div.ls img {height: 96px; width: 128px; margin: 16px 0;}
div.pt img {height: 128px; width: 96px; margin: 0 16px;}
div.pic ul {display: none;}
```

2.5 Die Galerieansicht

So interessant das vorige Stylesheet auch sein mag, es hat den Nachteil, dass die Namen der Fotos nicht angezeigt werden. Es mag eine perfekte Präsentation für den Künstler selbst sein, der wahrscheinlich alle Namen kennt und sie einfach nur in kompakter Form sehen will. Aber für Besucher ist das kaum hilfreich. Darum wollen wir nun unsere Dias in eine Fotogalerie verwandeln, bei der alle Bilder eine Beschriftung haben.

2.5.1 Die Dia-Styles entfernen

Um das Feld aufzuräumen (sozusagen), werden wir die Styles hinauswerfen, die die Hintergrundbilder an Ort und Stelle bringen (siehe Abbildung 2.7). Also bleibt uns das Stylesheet aus dem Listing 2.2.

Listing 2.2 Stylesheet-Ausschnitt

```
body {background: #EED; margin: 1em;}
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px;}
div.pic img {border: 1px solid; border-color: #444 #AAA #AAA #444;}
div.ls img {height: 96px; width: 128px; margin: 16px 0;}
div.pt img {height: 128px; width: 96px; margin: 0 16px;}
div.pic ul {display: none;}
```

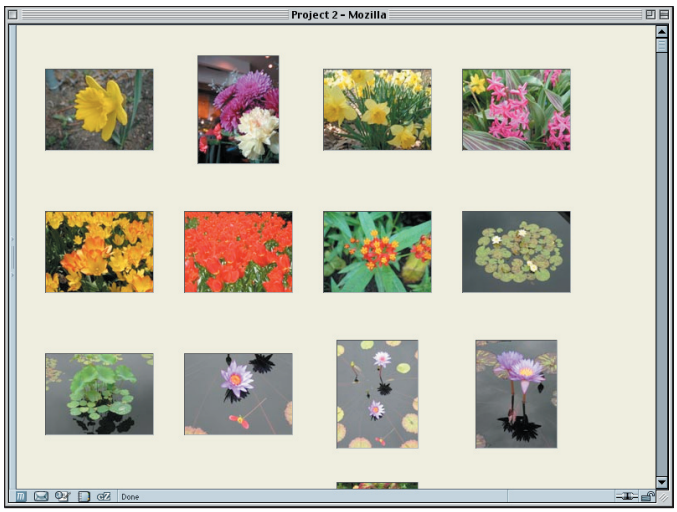


ABBILDUNG 2.7
Weg mit den Diarahmen

Bei diesem Stylesheet sind wir fast schon am Ziel angelangt. Als nächsten Schritt werden wir die Namen anzeigen.

2.5.2 Namentliche Offenbarungen

Damit das passiert, müssen wir die `ul`-Elemente offenlegen und die Teile, die wir an diesem Punkt nicht sehen wollen, verstecken. Der Teil des Aufdeckens ist ganz einfach: Wir entfernen `display: none;` aus der Regel `div.pic ul`. Natürlich bleibt uns da ein leerer Deklarationsblock übrig:

```
div.pic ul {}
```

Das ist schon in Ordnung, auch wenn's ein wenig nutzlos ist: Die Elemente werden ausgewählt, aber keine Styles angewendet. Also füllen wir die Leere mal auf. Wir geben Padding und Ränder sowohl für den `ul` als auch seine Font-Styles an.

```
div.pic ul {margin: 0.25em 0 0; padding: 0;
  font: bold small Arial, Verdana, sans-serif;}
```

Jetzt wird die Liste wieder insgesamt sichtbar. Weil wir nur wollen, dass die Namen erscheinen, werden wir die anderen Elemente aus der Liste verstecken (siehe Abbildung 2.8).

```
div.pic ul {margin: 0.25em 0 0; padding: 0;
  font: bold small Arial, Verdana, sans-serif;}
li.catno, li.price {display: none;}
</style>
```

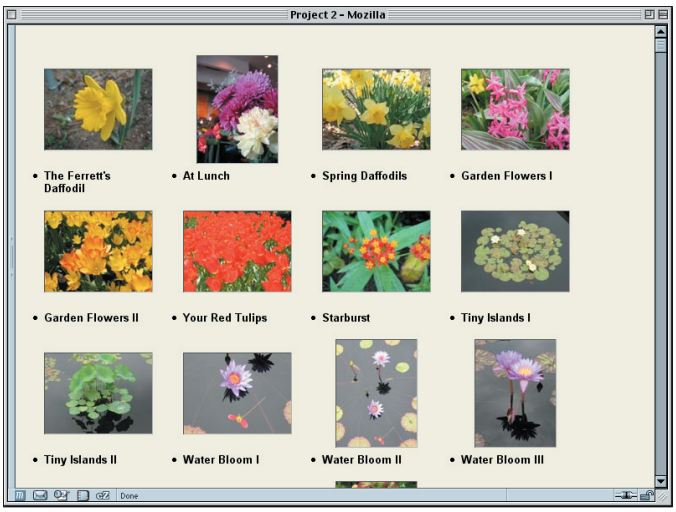


Abbildung 2.8

Der Name wird aufgedeckt, der Rest bleibt versteckt.

Gar nicht mal schlecht, aber da ist noch was zu tun. Wir wollen die Gliederungspunkte (Bullets) loswerden, weil sie nur ablenken und hässlich aussehen. Wir könnten sie einfach über die Eigenschaft `list-style` entfernen, aber andererseits könnten wir auch einfach das Element `li` ändern, damit sie kein Listenelement mehr sind. Das klappt auch prima, außer dass beim IE/Win die Bullets erhalten bleiben, also müssen wir beides machen. Da wir gerade dabei sind, zentrieren wir auch noch den Text.

```
div.pic ul {margin: 0.25em 0 0; padding: 0;
  font: bold small Arial, Verdana, sans-serif;}
li.title {display: block; list-style: none; text-align: center;}
li.catno, li.price {display: none;}
```

Jetzt werden wir die Namen ein wenig höher dichter an die Bilder schieben. Erinnern Sie sich noch, dass wir den Bildern Ränder hinzugefügt haben, als wir das Stylesheet für die »Dias« im vorigen Abschnitt erstellt haben? Dieser Rand schiebt die Namen von den querformatigen Bildern weg.

Achten Sie darauf, dass ich von den *Quer*formaten spreche. Die Hochformate haben keine oberen oder unteren Ränder. Darum brauchen wir nur die Ränder an den Querformaten zu ändern. In Abbildung 2.8 sehen Sie noch etwas Interessantes: Die ersten Zeilen der Bildunterschriften in jeder Reihe sind alle auf der gleichen Höhe. Das ist ein erhaltenswerter Effekt, also werden wir, statt einfach den unteren Rand bei den Querformaten zu entfernen, einfach den gesamten Rand nach oben verlegen.

```
div.ls img {height: 96px; width: 128px; margin: 32px 0 0;}
```

So weit, so gut, aber es gibt etwas, was wir noch nicht berücksichtigt haben: die Höhe der `div`s. Sie hängen immer noch bei `130px` Höhe plus Padding fest. Und zusammen heißt das, die Bildunterschriften laufen möglicherweise aus den `div`s heraus. Wenn wir den `div`s einen dünnen Rahmen hinzufügen, können wir sehen, was passiert, ohne den bisher erreichten Prozess zu sehr zu beeinträchtigen (siehe Abbildung 2.9).

```
div.pic {float: left; height: 130px; width: 130px;
  padding: 15px; margin: 5px 3px; border: 1px dotted silver;}
```

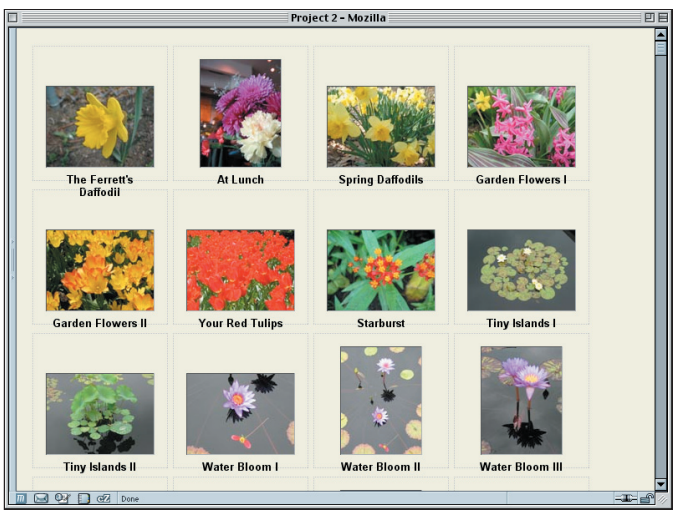


Abbildung 2.9

Wir haben es fast geschafft, aber es gibt noch ein paar Auswüchse.

Ein Blick auf den Explorer

So wie vorhin schon werden die `height`-Bugs im Explorer für Windows zu einem anderen Ergebnis als dem aus Abbildung 2.9 führen.

2.5.3 Aufräumen

An diesem Punkt brauchen wir nur noch die Höhe der `div`s heraufzusetzen, damit der Text genug Platz hat. Wieso machen wir das? Ein Grund dafür ist, dass das Layout im Internet Explorer für Windows zerschossen wird, wenn wir uns darum nicht kümmern. Ein anderer Grund liegt darin, dass es wahrscheinlich zu Überlappungen kommt, wenn über einem hochformatigen Bild ein langer Name erscheint. Ein dritter Grund ist, dass es einfach unordentlich ist, Elemente absichtlich aus ihren übergeordneten Elementen herausragen zu lassen (jedenfalls momentan – in einer späteren Phase des Projekts werden wir uns genau dieses Verhalten zunutze machen).

Wir werden also oben und unten das Padding der `div`s entfernen und `height` aufstocken. Ach, und dieser gepunktete Rahmen kann auch gleich weg.

```
div.pic {float: left; height: 190px; width: 130px;
  padding: 0 15px; margin: 5px 3px;}
```

Mit dieser Änderung sind wir bei unserem Stylesheet für die Galerieansicht angekommen (siehe Listing 2.3 und die Anzeige in Abbildung 2.10).

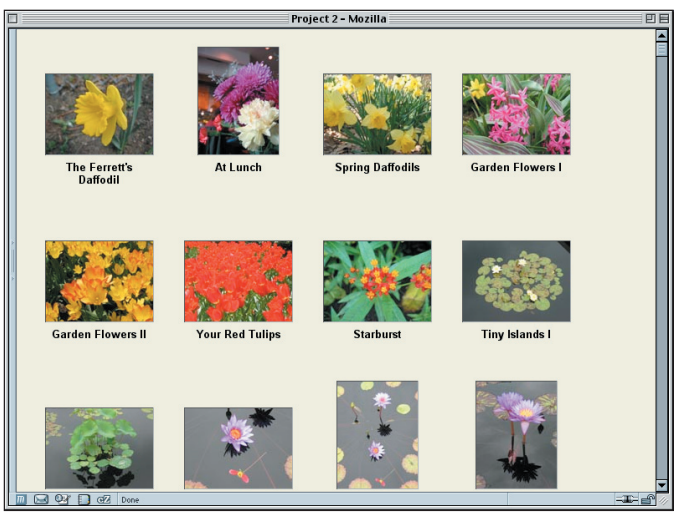
Jetzt ist ein guter Zeitpunkt, Ihre Arbeit in einer separaten Datei abzuspeichern, denn wir werden das Meiste von dem, was wir bisher gemacht haben, über Bord werfen und weitgehend neu anfangen.

Listing 2.3 Das Stylesheet der Galerieansicht

```
body {background: #EED; margin: 1em;}
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic {float: left; height: 190px; width: 130px;
  padding: 0 15px; margin: 5px 3px;}
div.pic img {border: 1px solid; border-color: #444 #AAA #AAA #444;}
div.ls img {height: 96px; width: 128px; margin: 32px 0 0;}
div.pt img {height: 128px; width: 96px; margin: 0 16px;}
div.pic ul {margin: 0.25em 0 0; padding: 0;
  font: bold small Arial, Verdana, sans-serif;}
li.title {display: block; text-align: center;}
li.catno, li.price {display: none;}
```

Abbildung 2.10

Eine Galerie der Blütenpracht



2.5.4 Höhen und Tabellen

Wir haben einen Wert von 190px gewählt, damit die Bildunterschriften genug Platz haben, auch wenn sie über drei Zeilen laufen, aber hier lauert eine potenzielle Fehlerquelle. Angenommen, der Benutzer ändert die Textgröße oder eine Bildunterschrift geht über fünf oder sechs Zeilen. Wenn so etwas passiert, wird das Layout total zerschossen – ohne dass man es wieder reparieren kann.

Diese Begrenzung ist bei als Floats definierten Elementen unvermeidbar. Im Wesentlichen steht jedes Element ganz für sich selbst allein: Seine Dimensionen sind völlig von allen anderen Elementen losgelöst. Anders gesagt: Die einzigen Elemente im gesamten Webdesign, die sich in der Größe nach ihren Nachbarn richten, sind Tabellenzellen.

Das bedeutet, dass es Situationen gibt, in denen eine Tabelle das beste Layout für eine Fotogalerie darstellt. Wenn es bei Ihnen so sein kann, dass Ihre Bildunterschriften

unvorhersehbar lang sind oder wenn Sie einfach nur wollen, dass jedes Bild in einer Box sitzt, die so hoch ist wie die höchste aus der Reihe, dann müssen Sie eine Tabelle verwenden. Sie können natürlich die Inhalte dieser Tabelle mit CSS und vielen der von uns hier erforschten Techniken stylen.

Eine andere sinnvolle Verwendung für eine Tabelle ist, wenn Sie eine bestimmte Anzahl von Bildern pro Reihe haben wollen, keins mehr oder weniger. Sie können die Dokumentstruktur natürlich auch mit CSS zerhacken, um das zu erreichen. Schauen Sie sich beispielsweise einmal dieses Raster an:

```
<div class="row">
  <div class="pic ls">...</div>
  <div class="pic pt">...</div>
  <div class="pic ls">...</div>
  <div class="pic ls">...</div>
</div>
```

Indem wir jeweils vier Bilder gruppieren, stellen wir sicher, dass es immer nur vier Bilder pro Reihe gibt. Aber warum sollten wir es uns schwer machen, wenn wir genau so gut eine Tabelle nehmen könnten? Das von uns vorgeschlagene Markup ist praktisch schon eine Tabelle. Schauen Sie es sich mal an:

```
<tr>
  <td class="pic ls">...</td>
  <td class="pic pt">...</td>
  <td class="pic ls">...</td>
  <td class="pic ls">...</td>
</tr>
```

Das Tabellen-Markup ist in diesem Fall einfacher, erfordert weniger Zeichen und wird ganz schlicht die Anzahl der Bilder pro Reihe begrenzen. Das ist eines der Dinge, die Tabellen machen können. Denken Sie allerdings doch daran, dass unsere ursprüngliche Absicht war, die Bilder »fließen« zu lassen, so dass jede Reihe so viele Bilder wie möglich enthalten kann, egal wie breit oder schmal das Fenster wird. Tabellen können das nicht, wohl aber Floats. Also nehmen Sie wie immer das Tool, das für die Arbeit am geeignetsten ist, und setzen es optimal ein.

Tabellen-Layout ohne Tabelle!

Es gibt tatsächlich eine Alternative für Floats und Tabellen: Nehmen Sie das hier gezeigte Markup und weisen Sie den `divs` Werte für Tabellen-`display` zu. Wir könnten zum Beispiel die Deklaration `div.row {display: table-row;}` und `div.row div {display: table-cell;}` festlegen. So wird eine tabellenähnliche Struktur aus Elementen, die keine Tabellen sind, erstellt, und es funktioniert sogar in aktuellen Browsern – außer (natürlich) im Explorer.

2.6 Die Katalogansicht

Wir haben uns damit beschäftigt, Bilder in Rastern »fließen« zu lassen, und wollen uns nun einem anderen Ansatz zuwenden. Dieses Mal werden wir eine vertikale Auflistung von Bildern einrichten und daneben den Namen, die Katalognummer und den Preis des Bildes setzen. Dafür werfen wir fast alle unsere Styles über Bord und fangen neu an. Wir behalten bloß die Styles für Body und Footer (siehe Listing 2.4). Wir sind im Grunde wieder da, wo wir mit dem Projekt angefangen haben (ziehen Sie noch mal Abbildung 2.1 zu Rate).

Listing 2.4 Ein (fast) kompletter Neuanfang

```
body {background: #EED; margin: 1em;}
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
```

2.6.1 Schon wieder Schwimmen

Um zu verhindern, dass das Layout über die Maßen lang wird, werden wir den Namen und die anderen Informationen nicht unter jede Abbildung setzen, sondern daneben. Text neben ein Bild zu stellen ist leicht: Sie lassen das Bild mit einem Float »schwimmen«. Außer dass wir in diesem Fall den Link, der das Bild beinhaltet, zum Float machen.

```
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic a.tn {float: left;}
</style>
```

Weil der Link ein Float ist, wird das Bild mitkommen, so wie Text mit dem Float in einem `div` mitkommt. In beiden Fällen wird der Inhalt in den Float gesetzt.

Wir werden den `div`s auch ein wenig Styling angedeihen lassen, nur ein kleiner Rand und ein Rahmen, damit wir sehen können, wie das Layout voranschreitet. Das Ergebnis können Sie in Abbildung 2.11 sehen.

```
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic {margin: 10px; padding: 0; border: 1px dotted gray;}
div.pic a.tn {float: left;}
```

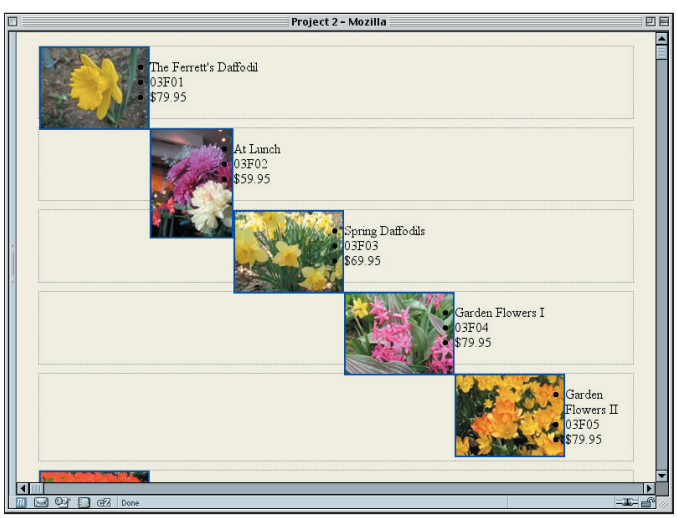


ABBILDUNG 2.11

Floats können gefährlich sein, wenn Sie nicht aufpassen.

Hoppla! Wenn wir das mit diesem Layout vorgehabt hätten – ganz klasse, kein Problem, aber *so* war es absolut nicht gemeint.

Was ist passiert? Genau das, was wir angegeben haben. Wenn Sie etwas »floaten« lassen, wird es aus dem normalen Fluss herausgenommen, und das heißt, es richtet sich nicht nach der Höhe der Vorfahr-Elemente. Darum ist jedes `div` genauso hoch wie sein normaler Flow-Inhalt (in diesem Fall die Informationsliste).

Und Floats beginnen dort, wo sie anfangen würden, wenn sie im normalen Fluss wären, und dann »schwimmen« sie auf die Seite. In diesem Fall »floaten« wir nach links. Also hat das zweite verknüpfte Thumbnail beim `div` ganz oben mit dem Floating angefangen und sich dann nach links ausgerichtet. Bevor es den linken Rand des Layouts erreicht hat, traf es auf einen anderen Float (die erste Verlinkung), also hat es Halt gemacht und sich selbst rechts von diesem anderen Float gesetzt. Das Gleiche passierte mit dem dritten, dem vierten usw.: Alle sind »gefloatet« und haben rechts von der Verlinkung angehalten, die davor kam.

Es gibt verschiedene Möglichkeiten, das zu reparieren. Wenn wir wollen, dass die `div`s der Bilder sich um die verknüpften Floats »herumziehen«, könnten wir auch die `div`s floaten. In aktuellen Browsern (und CSS2.1) wird ein Float-Element sich vergrößern, um alle Float-Nachkömmlinge zu umschließen. Das wollen wir hier aber nicht, weil die gepunkteten Rahmen später entfernt werden sollen. Sie sind hier nur zu Diagnosezwecken.

Also führen wir für jedes `div` ein `clear` aus. Wir werden auch die Breite der `div`s einstellen, weil wir wollen, dass sie später beschränkt werden.

```
div.pic {margin: 10px; padding: 0; border: 1px dotted gray;
  clear: left; width: 350px;}
```

Das wird alle `div`s unter alle linksgerichteten Floats schieben, die im Layout davor kommen – wie beispielsweise verknüpfte Floats (siehe Abbildung 2.12.).

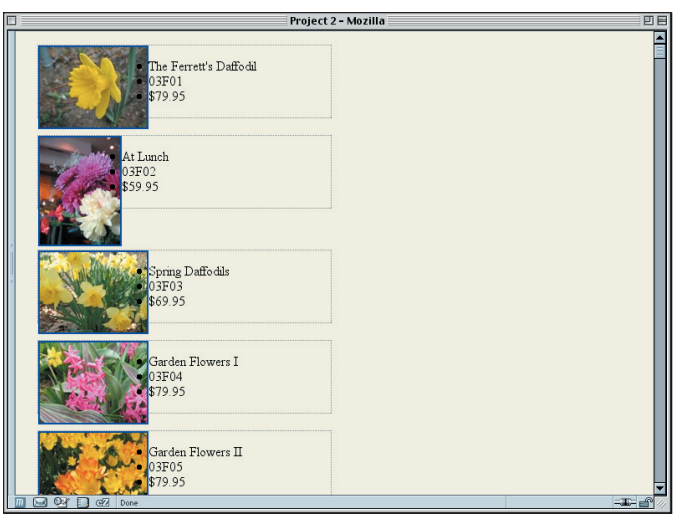


ABBILDUNG 2.12

Probleme mit Floats werden ausgeräumt.

Wie war das doch gleich?

Diese Erklärung könnte für Sie verwirrend klingen; wenn das der Fall ist, versuchen Sie, alles ein paar Mal ganz langsam zu lesen. Während Sie das machen, denken Sie daran, was passiert, wenn Sie einige Bilder zu Floats machen, neben denen kein Text steht. Im Prinzip passiert hier genau das Gleiche, außer dass der Text die Sache verkompliziert.

2.6.2 Ausrichtung und Platzierung

Wenn wir uns Abbildung 2.12 anschauen, sehen wir, dass wir uns noch um etwas kümmern müssen: Das hochformatige Bild ist nach links ausgerichtet. In einer Tabelle wäre das Bild wahrscheinlich in seiner eigenen Zelle, und wir bräuchten es bloß rechts auszurichten, aber diesen Luxus haben wir hier nicht. Zum Glück können wir einfach am Rand der verlinkten Floats feilen, um den gewünschten Effekt zu erzielen. Wir machen die Breite eben so, dass sie zum Bild passt, das es enthält, und stellen den linken Rand so ein, dass die Differenz zwischen seiner Breite und der Breite der querformatigen Verlinkungen ausgeglichen ist.

```
div.pic a.tn {float: left;}
div.pt a.tn {width: 96px; margin-left: 32px;}
</style>
```

Weil wir explizit die Breite der hochformatigen Verknüpfungen gesetzt haben, können wir das eigentlich nun auch für die querformatigen tun. Das tut nicht weh und könnte für die Zukunft auch ganz praktisch sein.

```
div.pt a.tn {width: 96px; margin-left: 32px;}
div.ls a.tn {width: 128px;}
</style>
```

Da wir gerade an den Bildern arbeiten, könnten wir diese hässlichen blauen Rahmen auch durch etwas weniger Aufdringliches ersetzen.

```
div.ls a.tn {width: 128px;}
a.tn img {border: 1px solid #333; border-width: 1px 2px 2px 1px;}
</style>
```

Wenn wir einen dickeren rechten und unteren Rahmen wählen, bekommt jedes Bild einen kaum wahrnehmbaren »Schlagschatten«-Effekt. Das können Sie bei einigen der Bilder in Abbildung 2.13 sehen.

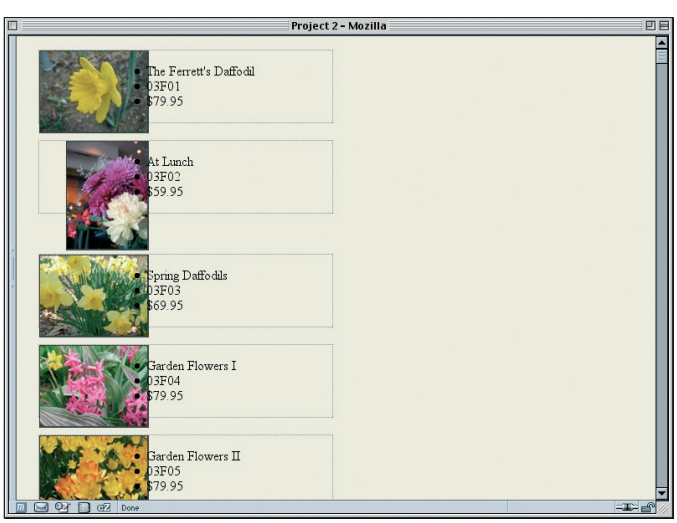


ABBILDUNG 2.13

Die Bilder werden ausgerichtet und erhalten durch Rahmen einen Schlagschatten.

Wenn Sie sich die Abbildung 2.13 genau anschauen, erkennen Sie zwei Sachen. Erstens überlappen sich der Text und die Rahmen der Bilder ein wenig. Das liegt daran, dass dank der von uns gerade hinzugefügten Rahmen die Bilder nun breiter sind als die verknüpften Floats, in denen sie stehen. Um das auszugleichen, sollten wir ihre Breite erhöhen. Ach was soll's, wir pumpen sie noch weiter auf, bis sie breiter als die Bilder sind.

```
div.pt a.tn {width: 100px; margin-left: 32px;}
div.ls a.tn {width: 132px;}
```

Zweitens können Sie sehen, dass die Aufzählungspunkte in die Bilder hineinragen. Das liegt an der Weise, wie Floats funktionieren, und der Art, wie Aufzählungspunkte (»Bullets«) platziert werden – und das passiert auch nicht in allen Browsern. Wenn wir diese Bullets in der Liste erhalten wollten, hätten wir nun ein Problem, aber wir werden sie sowieso wegnehmen, also macht das nichts. Das können wir daher gleich mal erledigen.

```
a.tn img {border: 1px solid #333; border-width: 1px 2px 2px 1px;}
div.pic li {list-style: none;}
</style>
```

Das ist gut, um Überlappungen zu verhindern, aber der Text klebt immer noch zu eng an den verknüpften Floats. Wir schieben den Text ein wenig zur Seite, indem wir eine Kombination aus Rändern und Padding nutzen. Wir geben einen linken Rahmen hinzu, damit wir sehen können, wo genau der Rahmen der Liste aufhört. Das Ergebnis können Sie in Abbildung 2.14 betrachten.

```
a.tn img {border: 1px solid #333; border-width: 1px 2px 2px 1px;}
div.pic ul {margin: 0 0 0 140px; padding: 0 0 0 0.5em;
  border-left: 1px solid;}
div.pic li {list-style: none;}
```

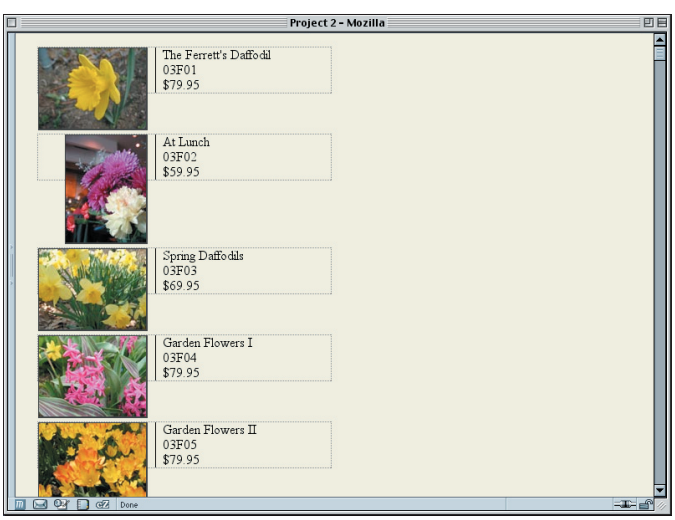


Abbildung 2.14

Ränder und Padding helfen, den Text von den verknüpften Bildern zu trennen.

Wohin mit den Punkten?

Aufzählungspunkte werden in Relation zur linken Kante vom Inhalt des Listen-Elements angeordnet; egal wo diese Kante landet, der Punkt ist ein kleines Stück weiter weg. Weil die linken Kanten der Listen-Elemente genau mit den Float-Bildern abschließen, landen die Punkte direkt auf den Bildern.

Wie hat das funktioniert? Der linke Rand von 140 Pixel rutscht tatsächlich »unter« den Link im Float und dehnt sich bis zur linken Kante des `div`s aus. Wenn Sie sich das zweite `div` in Abbildung 2.14 anschauen, können Sie sehen, dass das `div` bis zum rechten Rand des Links geht. Tatsächlich passiert das bei allen `div`s, aber es fällt nur bei den hochformatigen Verknüpfung auf.

2.6.3 Die Listings werden verbessert

Jetzt, wo wir die Bilder und Links alle ausgerichtet und die Listen so angeordnet haben, dass sie ein Stück von den Bildern getrennt sind, wollen wir unsere Aufmerksamkeit dem Text innerhalb dieser Listen zuwenden. Als Erstes wollen wir diese Rahmen loswerden, weil sie langsam im Weg stehen. Darum sollte die Regel für `div.pic` nun wie folgt aussehen:

```
div.pic {margin: 10px; padding: 0;
  clear: left; width: 350px;}
```

Entsprechend sollte die Regel für `div.pic ul` wie folgt geändert werden:

```
div.pic ul {margin: 0 0 0 140px; padding: 0 0 0 0.5em;}
```

Jetzt, wo das fertig ist, wollen wir die Namen der Bilder hervorheben. Es wäre schön, wenn sie einen fetten Sans-Serif-Font und auch einen unteren Rahmen haben. Und sie sollten auch ein wenig vom oberen Rand der als Floats definierten Links nach unten verschoben sein (also etwa ein halbes em).

```
div.pic li {list-style: none;}
div.pic li.title {font: bold small Arial, Verdana, sans-serif;
  padding-top: 0.5em; border-bottom: 1px solid;}
</style>
```

Die Schriftgröße haben wir hier auf `small` gestellt, weil die Sans-Serif-Schriften besser ausschauen, wenn sie ein wenig kleiner als normal sind. Damit alles einheitlich wird, werden wir auch die Fonts der Listenelemente auf klein stellen.

```
div.pic li {list-style: none; font-size: small;}
```

Und jetzt zur Katalognummer und dem Preis. Diese sind nicht so wichtig wie der Name, also lassen wir sie in den Hintergrund übergehen, indem wir eine dunklere Schattierung wählen. Wir werden sie auch nach rechts ausrichten und den Preis kursiv setzen, um ihn ein wenig zu betonen (siehe Abbildung 2.15).

```
div.pic li.title {font: bold small Arial, Verdana, sans-serif;
  padding-top: 0.5em; border-bottom: 1px solid;}
div.pic li.catno {color: #776; text-align: right;}
div.pic li.price {color: #776; text-align: right;
  font-style: italic;}
</style>
```

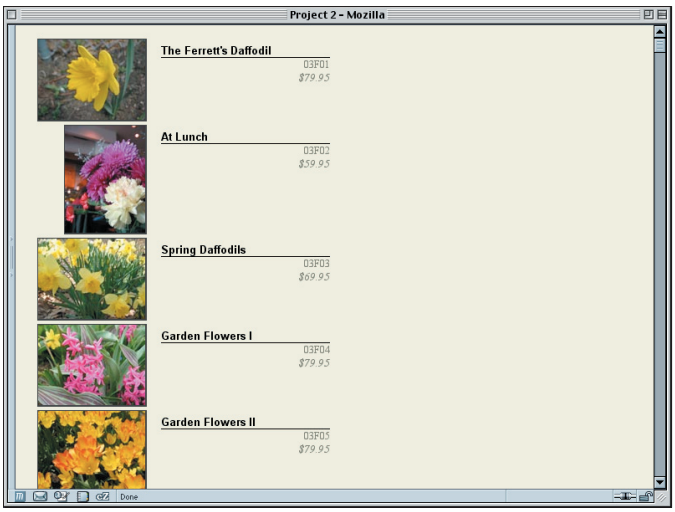


Abbildung 2.15

Der Text sieht dank Ausrichtung und Farbe schon deutlich besser aus.

Hier könnten wir im Prinzip schon aufhören, aber wir machen noch weiter. Es wäre schon ganz cool, wenn Katalognummer und Preis nebeneinander stehen, oder? Na, vielleicht ist cool nicht der richtige Ausdruck, aber wohl doch eleganter. Auf jeden Fall ist das jetzt dran. Und wir machen es, ohne weitere Floats in das Layout einzubauen.

Damit das funktioniert, werden wir den Preis um eine Zeilenhöhe hochschieben. Damit *das* wiederum funktioniert, müssen wir sicherstellen, dass die Zeilenhöhe in allen Browsern gleich ist und die Katalognummer dabei nicht im Weg steht. Zuerst werden wir die Höhe der Listenelemente festlegen, indem wir `line-height` explizit definieren und Ränder und Padding auf Null setzen.

```
div.pic li {list-style: none; font-size: small;
  line-height: 1.2em; margin: 0; padding: 0;}
```

Jetzt schieben wir die Katalognummer zur Seite. Das machen wir, indem der rechte Rand auf 4,5 em eingestellt wird.

```
div.pic li.catno {color: #776; text-align: right;
  margin-right: 4.5em;}
```

Jetzt brauchen wir nur noch den Preis in den Bereich hochzuschieben, den wir gerade dafür erstellt haben. Wissen Sie noch den Wert für `line-height`? Wir werden einfach einen negativen oberen Rand setzen, der genau diesen Wert hat, und der Text wird an die richtige Stelle rutschen (siehe Abbildung 2.16).

```
div.pic li.price {color: #776; text-align: right;
font-style: italic; margin: -1.2em 0 0 0;}
```

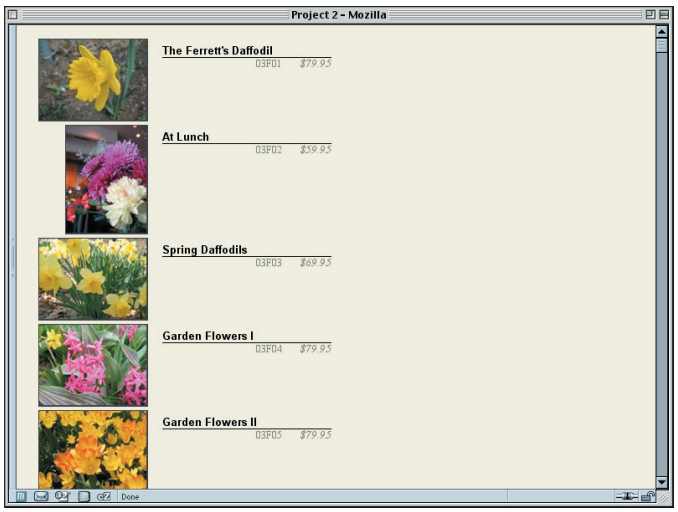


Abbildung 2.16

Durch das Wunder der Ränder steht Text neben Text.

Ein Blick auf den Explorer

Vom Rest dieses Projekts wird praktisch nichts im Internet Explorer 5.x für Windows funktionieren, weil dieser `margin-auto` nicht versteht. Der erste Teil dieses Abschnitts klappt im IE6 aber ganz gut, solange Sie sich im Standardmodus befinden.

2.6.4 Informationen im Kasten

Wir machen mit unserem fortgeschrittenen Layout weiter, indem wir einen vertikalen Rahmen links von der Katalognummer setzen, einen anderen zwischen Nummer und Preis und dann den ganzen Schwung Textinformation in einen Kasten packen.

Damit die Rahmen dort stehen, wo wir sie haben wollen, werden wir sehr ähnliche Styles auf die Katalog- und Preis-Listenelemente anwenden. Wir werden im Grunde allen eine Breite zuweisen und sie dann soweit wie nötig nach rechts schieben. Das machen wir, indem wir den linken Rand auf `auto` stellen und den Rest auf Null. Also legen wir für den Preis fest:

```
div.pic li.price {color: #776; text-align: right;
  font-style: italic; margin: -1.2em 0 0 auto;
  width: 4em; border-left: 1px solid;}
```

Dann ersetzen wir das `margin-right` der Katalognummer mit einer `margin`-Deklaration und geben ihr die gleiche Breite und den gleichen Rahmen wie dem Preis. Schauen Sie sich das Ganze in Abbildung 2.17 an.

```
div.pic li.catno {color: #776; text-align: right;
  margin: 0 4.5em 0 auto;
  width: 4em; border-left: 1px solid;}
```

Wenn wir jetzt den Kasten um diese ganze Textinformation legen wollen, wird es etwas knifflig. Wenn wir dem `ul`-Element nur einen Rahmen und einen Hintergrund hinzufügen, kommt IE6 für Windows völlig durcheinander und lässt Elemente einfach so verschwinden. Ehe wir also das arme Schätzchen ganz durcheinander bringen, werden wir einfach die bösen Styles vor seinen Augen verstecken. Und das geht so:

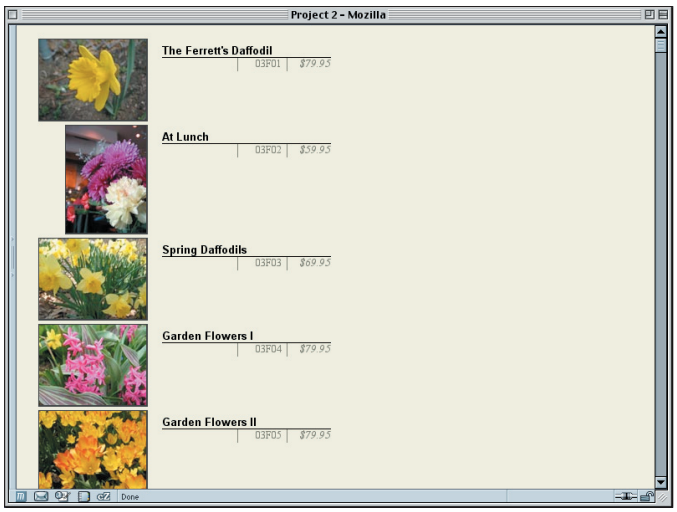


ABBILDUNG 2.17

Mehr Rahmen durch weitere Ränderwunder

```
div.pic ul {margin: 0 0 0 140px; padding: 0 0 0 0.5em;}
html>body div.pic ul {background: #CCB; border: 3px double #552;}
div.pic li {list-style: none; font-size: small;
  line-height: 1.2em; margin: 0; padding: 0;}
```

Der erste Teil dieses Selektors (der mit `html>body`) versteckt die Regel vor IE/Windows. Es ist ein Kind-Selektor und perfekt gültiges CSS2, aber der IE/Windows begreift es einfach nicht. Fähigere Browser aber doch, und die können diese Styles sehen und anwenden.

Durch das Hinzufügen der Rahmen müssen wir den Preis ein wenig nach links verschieben, damit er nicht direkt neben dem doppelten Rahmen steht. Das erledigen wir mit einer kleinen Änderung seines rechten Randes.

```
div.pic li.price {color: #776; text-align: right;
  font-style: italic; margin: -1.2em 3px 0 auto;
  width: 4em; border-left: 1px solid;}
```

Zum Schluss schaffen wir noch etwas Platz zwischen jedem Listing in unserer Katalogansicht. Dafür haben wir folgende Auswahl: Wir können den als Floats definierten Links oder den Bildern innerhalb dieser Verlinkungen einen unteren Rahmen hinzufügen. Wegen der Art, wie `clear` definiert ist, würde es nicht funktionieren, wenn wir den `divs` einen oberen Rand hinzufügen. Um ein paar obskure Bugs im Explorer zu vermeiden, werden wir den Rand auf das Bild setzen.

```
a.tn img {border: 1px solid #333; border-width: 1px 2px 2px 1px;
  margin: 0 0 1em;}
```

Damit sind wir bei dem Stylesheet aus Listing 2.5 angekommen, das in Abbildung 2.18 gezeigt ist.

Clear geht vor

`clear` funktioniert so, dass der obere Rand des Elements, auf das sich `clear` bezieht, erhöht wird, bis seine äußere obere Rahmenkante sich genau unterhalb der unteren Kante der vorigen Float-Elemente befindet. Wenn wir also den oberen Rand auf die `divs` eingestellt hätten, wären unsere Einstellungen durch `clear` überschrieben worden.

Listing 2.5 Das komplette »Katalog«-Stylesheet

```
body {background: #EED; margin: 1em;}
div#footer {clear: both; padding-top: 3em;
  font: 85% Verdana, sans-serif;}
div.pic {margin: 10px; padding: 0;
  clear: left; width: 350px;}
div.pic a.tn {float: left;}
div.pt a.tn {width: 100px; margin-left: 32px;}
div.ls a.tn {width: 132px;}
a.tn img {border: 1px solid #333; border-width: 1px 2px 2px 1px;
  margin: 0 0 1em;}
div.pic ul {margin: 0 0 0 140px; padding: 0 0 0 0.5em;}
html>body div.pic ul {background: #CCB; border: 3px double #552;}
div.pic li {list-style: none; font-size: small;
  line-height: 1.2em; margin: 0; padding: 0;}
div.pic li.title {font: bold small Arial, Verdana, sans-serif;
  padding-top: 0.5em; border-bottom: 1px solid;}
div.pic li.catno {color: #776; text-align: right;
  margin: 0 4.5em 0 auto;
  width: 4em; border-left: 1px solid;}
div.pic li.price {color: #776; text-align: right;
  font-style: italic; margin: -1.2em 3px 0 auto;
  width: 4em; border-left: 1px solid;}
```

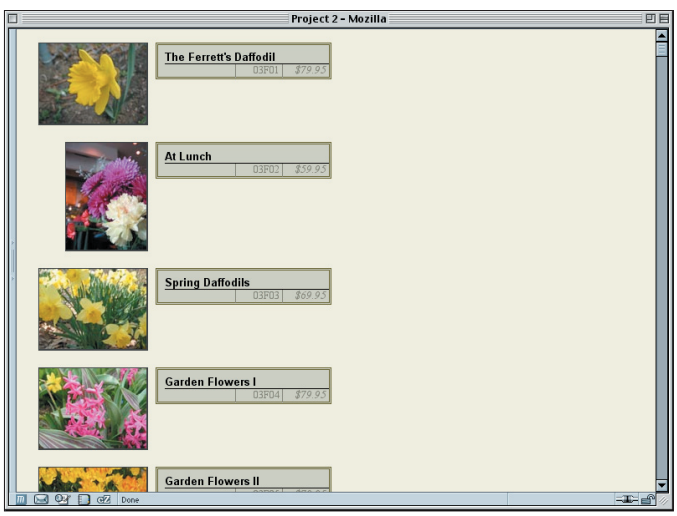


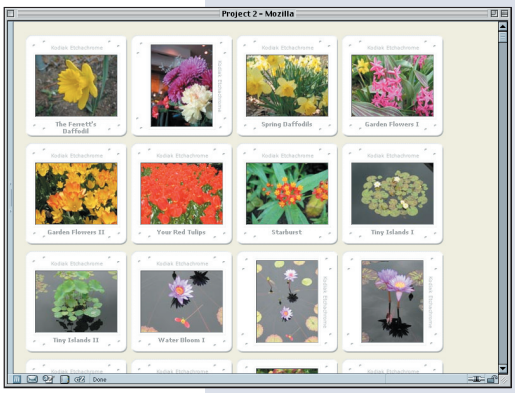
Abbildung 2.18

Die Informationen werden in einer Katalogansicht präsentiert.

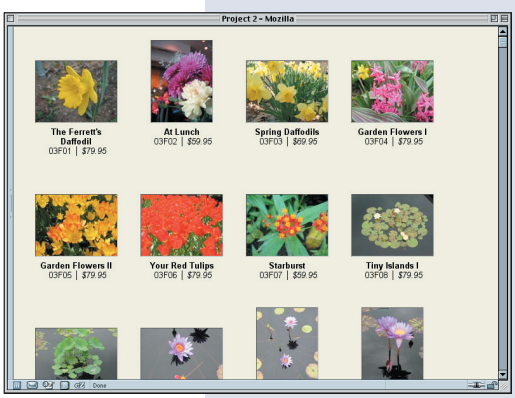
2.7 Spielwiese

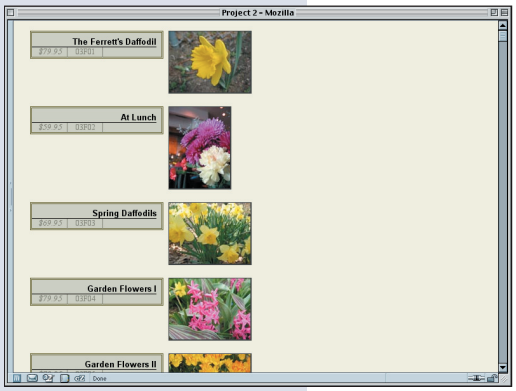
Versuchen Sie, die folgenden Änderungen an unserer Projektarbeit nachzubauen.

1. Setzen Sie in der Kontaktbogenansicht den Namen direkt unter das Bild, so dass es wirkt, als sei er auf den unteren Teil des Diarahmens geschrieben. Dafür werden Sie den unteren Rand der Bilder entfernen müssen, ohne das Gesamtlayout durcheinander zu bringen. Beachten Sie, dass dies für hochformatige Bilder nicht klappt, weil CSS Text nicht rotieren lassen kann, also müssen Sie Ihre Styles entsprechend beschränken. Denken Sie auch dran, dass lange Namen aus dem Diarahmen herauslaufen könnten, also wäre eine Eigenschaft wie `overflow` ganz praktisch.



2. Fügen Sie in der Galerieansicht die Katalognummer und den Preis wieder dem Layout hinzu, aber stellen Sie sie nebeneinander statt übereinander. Das wird die Galerieansicht verbessern, ohne das Layout über die Maßen zu ändern. Denken Sie daran, dass ein wenig mehr Höhe gebraucht werden könnte.





3. Versuchen Sie, in der Katalogansicht die ganze Geschichte umzudrehen, so dass die Bilder rechts und die Texte links sind. Das führt zu mehr als nur der Änderung der Float-Ausrichtung: Sie werden auch das Layout der Listenelemente anzupassen haben.

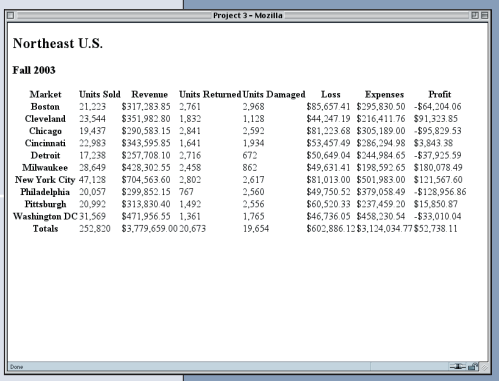
3

EIN FINANZBERICHT WIRD GESTYLT

> *Das Genie des Kapitalismus besteht gerade in seinem Mangel an Moral. Bevor er nicht reich genug ist, sich ein eigenes Orchester zu leisten, ist ein Kapitalist jemand, der es sich per Definition kaum erlauben kann, an etwas anderes zu glauben als die Doktrin der schwarzen Zahlen.*
>
> *Lewis H. Lapham*

Obwohl wir daran gewöhnt sind, das Web als eine Ansammlung von persönlichen Homepages, hochpotenten E-Commerce-Sites, Randgruppen und weitverstreuten Infobrocken zu sehen, ist da noch eine ganze Menge mehr los – auch wenn vieles davon nicht öffentlich ist. Unternehmen haben beispielsweise Intranets, die voll von HTML-basierten Informationen sind, und die reichen von Portalen für Angestellte bis zu Webmail-Oberflächen und darüber hinaus.

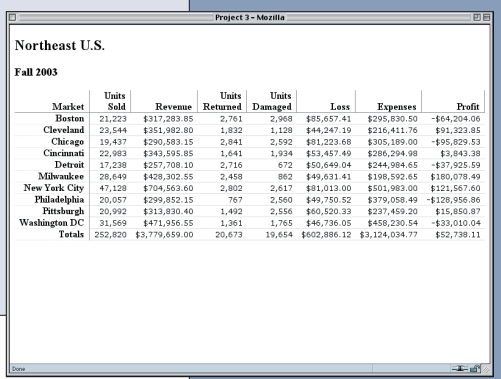
Obwohl ein ganzes Buch nur dem Thema gewidmet werden kann, wie gerade mal ein Ausschnitt dieser Informationen zu stylen ist, soll es für uns reichen, den Typ Information zu untersuchen, der wohl Hauptbestandteil von Firmensites ist: ein Finanzbericht, in dem Gewinne und Verluste eines bestimmten Quartals auf verschiedenen Märkten angezeigt werden. Wir werden nicht versuchen, das Tabellen-Markup durch `div`s und CSS zu ersetzen – das wäre töricht. Da unsere Daten schon in



Northeast U.S.

Fall 2003

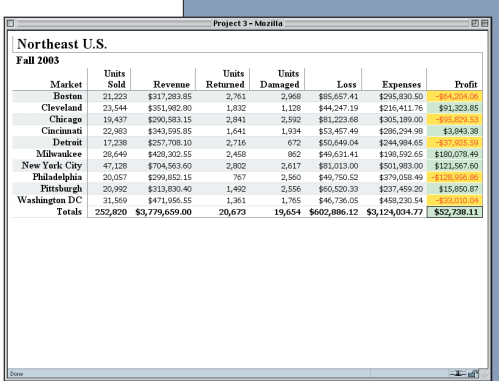
Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	**252,820**	**$3,779,659.00**	**20,673**	**19,654**	**$602,886.12**	**$3,124,034.77**	**$52,738.11**

einer Tabelle sitzen, werden wir auch ein Tabellen-Markup verwenden. Wir werden also der Struktur nützliche Hooks geben und diese dann verwenden, um die Tabelle mit ihren Inhalten zu stylen.

3.1 Projektziele

Unser Ziel für dieses Projekt ist, eine Tabelle mit Finanzdaten sowohl für die Ausgabe auf dem Bildschirm als auch für Print zu stylen und dafür das Markup so wenig wie möglich zu erweitern. Das werden wir hauptsächlich dadurch erreichen, dass wir den bereits vorhandenen Elementen Klassen- und ID-Hooks zur eindeutigen Referenzierung zuweisen.

Für dieses Projekt hat uns die Geschäftsführung detaillierte Designvorgaben gemacht:

- Die Summenzeile unten im Bericht soll visuell hervorgehoben werden, vorzugsweise durch fettgedruckte Zahlen.
- Die Überschriften (sowohl in der oberen Reihe als auch an der Seite des Berichts) sollen rechts ausgerichtet und von den Zahlen aus der Tabelle durch graue Linien getrennt sein. Die oberen Überschriften sollen durch vertikale graue Linien voneinander getrennt sein.
- In einem Browser sollen alle Zeilen des Berichts durch eine hellgraue Linie voneinander getrennt sein. Als Print soll jede zweite Zeile einen hellgrauen Hintergrund haben, und Spalten sollen durch vertikale hellgraue Linien voneinander getrennt sein.
- In einem Webbrowser soll jede negative Zahl rot mit einem gelben Hintergrund erscheinen, als Print soll sie kursiv ausgegeben werden.
- Die Erträge (die Spalte ganz rechts) sollen visuell hervorgehoben werden.
- Die Dollarbeträge sollen an der Dezimalstelle ausgerichtet sein, und alle anderen sollen sich an der Tausender-Kommastelle ausrichten.

Zusätzlich haben wir noch den vagen (aber nützlichen) Auftrag bekommen, dass »alles gut aussehen« soll. Mit diesen Zielen vor Augen bringen wir die Sache nun auf den Weg.

3.2 Vorbereitungen

Laden Sie die Dateien für Projekt 3 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch03proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und erneut laden.

3.3 Styling für den Bildschirm

Weil alle Anwender über einen Webbrowser auf die Berichte zugreifen, werden wir mit dem Erstellen von Screen-Styles anfangen. In dieser Phase lassen wir die Designziele für Print erst einmal beiseite und konzentrieren uns darauf, was auf dem Monitor ausgegeben wird.

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

3.3.1 Die Grundlagen

Als Basis unserer Arbeit werden wir eine Berichtstabelle verwenden, die die Datenbank-Mitarbeiter für uns erstellt haben. Ihr Markup ist, wie bereits von uns festgelegt, so schlicht und schnörkellos wie möglich. Die ersten paar Reihen der Tabelle werden im Listing 3.1 gezeigt, und die gesamte Tabelle in ihrer Standardansicht können Sie in Abbildung 3.1 sehen.

Listing 3.1 Der Anfang der Tabelle

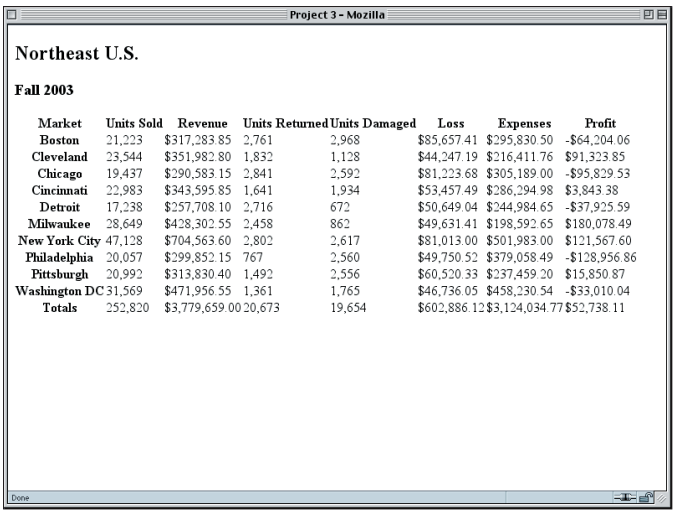
```
<table cellspacing="0" summary="Q3 Financial Results">
<thead>
<tr>
<th>Market</th>
<th>Units Sold</th>
<th>Revenue</th>
<th>Units Returned</th>
<th>Units Damaged</th>
<th>Loss</th>
<th>Expenses</th>
<th>Profit</th>
</tr>
</thead>
<tr>
<th>Boston</th>
<td>21,223</td>
<td>$317,283.85</td>
<td>2,761</td>
<td>2,968</td>
<td>$85,657.41</td>
<td>$295,830.50</td>
<td>-$64,204.06</td>
</tr>
<tr>
<th>Cleveland</th>
<td>23,544</td>
<td>$351,982.80</td>
<td>1,832</td>
<td>1,128</td>
<td>$44,247.19</td>
<td>$216,411.76</td>
<td>$91,323.85</td>
</tr>
```

Zellenabstand

Beachten Sie, dass hier das HTML-Attribut `cellspacing` vorkommt. Bei CSS2 wird die Trennung zwischen Zellen in einer Tabelle über die Eigenschaft `border-spacing` in Verbindung mit dem entsprechenden Wert für die Eigenschaft `border-collapse` erzwungen, aber der Support für diese Vorgehensweise ist ziemlich schlecht, also bleibt uns nur `cellspacing`. Das ist immer noch gültiges HTML (und sogar gültiges XHTML), also kein Problem.

ABBILDUNG 3.1

Der Bericht in seinem (weitgehend) ungestylten Zustand



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

Als Erstes fällt uns auf, dass die Inhalte der `th`-Elemente alle fett und zentriert sind. Das ist die Standarddarstellung des Browsers für `th`-Elemente und wird wie folgt repräsentiert:

```
th {font-weight: bold; text-align: center;}
```

Tatsächlich weisen viele marktübliche Browser (wie die, die auf Mozilla basieren) ein Standard-Stylesheet auf, das wahrscheinlich schon eine ganz ähnliche Regel wie diese enthält. Über diese Standard-Styles müssen wir uns hinwegsetzen, um unsere Designziele zu erreichen.

Aufschlussreiche Styles

Wenn Sie sich mal die Styles anschauen wollen, die von Gecko-basierten Browsern wie Mozilla, Firefox und Netscape 7.x verwendet werden, dann durchsuchen Sie Ihre Festplatte nach `html.css`. Wenn Sie mehr als einmal fündig werden, nehmen Sie die Datei mit dem neuesten Änderungsdatum.

Als zweite Sache fällt auf, dass hier `th`-Elemente vorkommen, die wir unterschiedlich stylen sollen, die aber scheinbar nicht zu unterscheiden sind. Also müssen wir auch etwas dafür tun, dass wir erkennen können, welche `th`-Elemente welche Rolle haben, damit wir sie dementsprechend stylen können.

Schließlich haben wir `h2`- und `h3`-Elemente, die vor der Tabelle kommen. Deren Styling verschieben wir, bis die Tabelle fertig ist. Auf diese Weise können wir sichergehen, dass die Styles der Header am Ende auf jeden Fall zum Layout der Tabelle passen werden.

3.3.2 Styles der schwarzen Zahlen

Unter Berücksichtigung unserer Designziele können wir ein paar Sachen vornehmen, die sich auf alle Zellen in der Tabelle auswirken, egal ob es Header oder Daten sind. Da ist beispielsweise die Anweisung, dass alle Spaltenüberschriften (hier in den `th`-Elementen enthalten) rechtsbündig sein sollen. Und alle Zahlen sollen ausgerichtet sein. Das können wir beides durch ein insgesamt rechtsbündiges Ausrichten auf einen Schlag erledigen (wenigstens bis zu einem gewissen Grad).

```
<style type="text/css" media="screen">
th, td {text-align: right;}
</style>
```

Das bringt schon ganz schön viel, aber einiges bleibt auch noch. Bei den Zahlen selbst werden die Punkte und Kommata ohne eine passende Schriftart nicht ganz ausgerichtet sein. Das liegt daran, dass in einem proportionalen Font wie Times die Breite der Zahlen immer ein wenig unterschiedlich sein kann. Damit die Punkte und Kommata alle passend stehen, müssen wir den Zahlen erst eine Schriftart zuweisen, die für jede Ziffer die gleiche Breite hat. Wir könnten dafür jeden beliebigen Monospace-Font nehmen, aber wir wählen stattdessen die allseits beliebte Verdana, weil ihre Zahlen-Glyphen alle die gleiche Breite aufweisen – und das ist nicht bei allen Sans-Serif-Fonts so. Falls der User keine Verdana verfügbar hat, werden wir ein paar Monospace-Alternativen anfügen.

```
<style type="text/css" media="screen">
th, td {text-align: right;}
td {font: smaller Verdana, "Andale Mono", Courier, "Courier New",
  monospace;}
</style>
```

Wir haben die Schriftgröße auf `small` gesetzt, weil Monospace- und Sans-Serif-Fonts dazu neigen, dem Auge größer zu erscheinen als Fonts mit Serifen. Durch die Wahl einer kleineren Schriftgröße haben wir diesem Effekt entgegengesteuert.

Der letzte Schritt in unseren »globalen« Styles ist nun, die Zellen seitlich noch mit etwas Padding aufzufüllen. Das verhindert, dass lange Zahlen ihren Nachbarn zu sehr auf den Pelz rücken.

```
th, td {text-align: right; padding: 0 0.5em;}
```

Schließlich wollen wir noch die Linien für die Zeilentrennung einstellen. Dafür werden wir eigentlich nur allen Zellen einen unteren Rahmen geben und das später dort außer Kraft setzen, wo es uns passend erscheint.

```
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
```

Wie wir in Abbildung 3.2 sehen können, richten sich die Zahlen schon ganz schön in den Spalten aus, und die Reihen werden durch Linien getrennt.

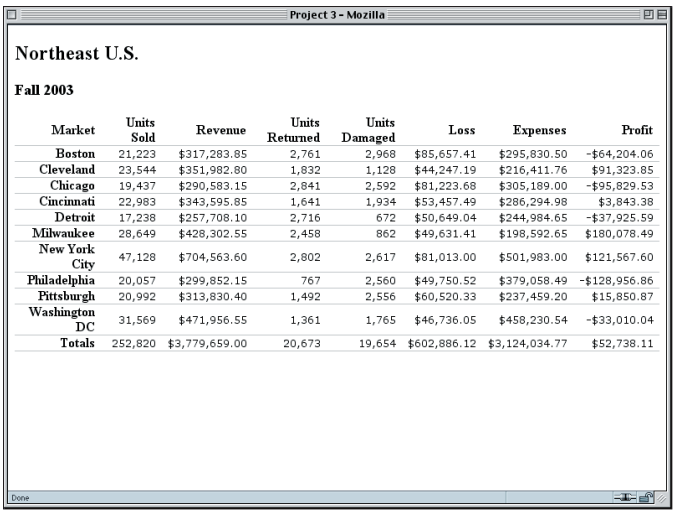
Ein paar Sachen liegen noch an: Einige der Reihen sind beispielsweise wegen der langen Städtenamen höher als andere.

Ausrichtung an Zeichen

Obwohl CSS2 die Möglichkeit eingeführt hat, Text an einem Zeichen auszurichten, wird dies noch nicht durch Webbrowser unterstützt, also müssen wir das ignorieren. Das ist wirklich eine Schande, denn die gewünschte Ausrichtung könnten wir mit einer Deklaration wie `text-align: "."` bekommen, womit sich die Zahlen an den Dezimalstellen ausrichteten.

Abbildung 3.2

Das Fundament ist gelegt.



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

3.3.3 Feinschliff bei den Tabellenüberschriften

Wir wollen uns nun den Spaltenüberschriften im Tabellenkopf zuwenden. Wir wissen, dass diese anders gestylt werden sollen als die Zeilenbezeichnungen auf der linken Seite. Um die eine Gruppe von `th`-Elementen von der anderen unterscheiden zu können, machen wir uns das `thead`-Element zunutze, das die Kopfzeile der Tabelle umschließt.

Damit können wir Styles direkt nur auf diese Zellen anwenden und keine andere ist betroffen.

Andere Ansätze

Tatsächlich gibt es bei den »oberen Überschriften« noch ein paar andere Dinge, die strukturell einmalig sind. Beispielsweise sind sie `th`*-Elemente, die von einer Tabellenspalte abstammen, die die ersten* `tr`*-Nachfahren des* `thead`*-Element sind. Wir könnten theoretisch die CSS2-Pseudoklasse* `:first-child` *benutzen, um sie auszuwählen. Das sähe dann etwa so aus:*

```
tr:first-child th {...styles for “top labels” here...}
th {...styles for other ‘th’ elements here...}
```

Das Problem ist aber, dass `:first-child` *nicht durch den Internet Explorer 6 unterstützt wird, was den Nutzen recht einschränkt. Es ist wirklich sinnvoller, hier* `thead` *zu verwenden, weil es sowieso schon da ist. Aber für die Fälle, bei denen Mozilla oder einer seiner Cousins der Intranet-Browser ist, kann die Pseudoklasse* `:first-child` *verwendet werden.*

Zuerst wollen wir den Text der Spaltenüberschriften vertikal ausrichten. Diese Headings sind normalerweise unten bündig ausgerichtet.

```
td {font: smaller Verdana, "Andale Mono", Courier, "Courier New",
  monospace;}
thead th {vertical-align: bottom;}
</style>
```

Wir müssen auch vertikale Linien hinzufügen, um die Spaltenüberschriften getrennt zu halten, und einen grauen Rahmen, um diese Überschriften von den Zahlen in den Datenzellen zu trennen.

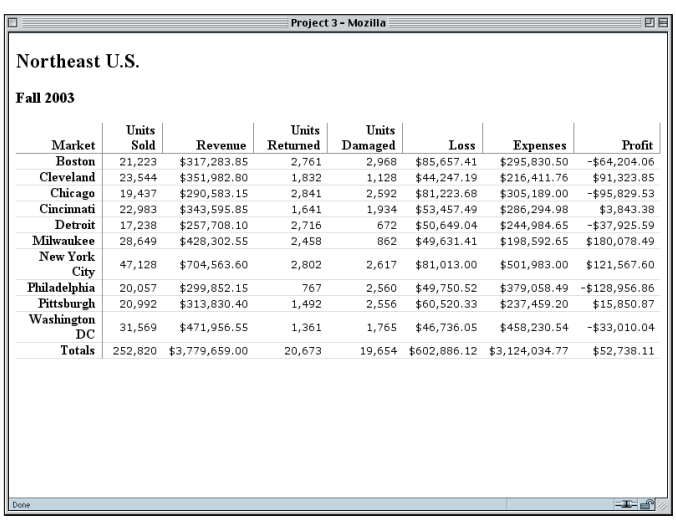
```
thead th {vertical-align: bottom; border: 1px solid gray;
  border-width: 0 1px 1px 0;}
```

Weil der Selektor `thead th` eine größere Spezifität hat als der Selektor `th`, werden durch die folgende Regel die Rahmen-Styles aus dieser Regel gegenüber denjenigen gewinnen, die wir vorher geschaffen haben:

```
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
```

Jetzt können wir bei den Beschriftungen auf der linken Seite ähnliche Schritte vornehmen. Für diese wollen wir eigentlich nur eine graue Linie haben, die die Zeilenbeschriftungen von den Datenzellen trennt. Das ist einfach genug, wie Sie aus Abbildung 3.3 sehen können.

```
thead th {vertical-align: bottom; border: 1px solid gray;
  border-width: 0 1px 1px 0;}
th {border-right: 1px solid gray;}
</style>
```



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

ABBILDUNG 3.3

Die Zeilen- und Spaltenüberschriften werden sichtbar voneinander getrennt.

Vertikale Varianten

Beachten Sie, dass sich `vertical-align` nur auf diese Weise verhält, wenn es auf Tabellenzellen angewendet wird. Für alle anderen Elemente hat es sehr unterschiedliche Effekte, weil es bei Nicht-Tabellenelementen dafür benutzt wird, wie weit ein Element bezogen auf die normale Grundlinie hoch- oder tiefgestellt wird.

Was ist Spezifität?

Spezifität ist die Gewichtung, die CSS einem bestimmten Selektor zuweist. Als Teil der Kaskade hilft Spezifität den Browsern dabei herauszufinden, welcher Style gelten soll, falls zwei Regeln im Konflikt stehen. Für Details schauen Sie bitte im Abschnitt 6.4 der CSS2.1-Spezifikation nach (`http://www.w3.org/TR/CSS21`).

Ein paar Sachen können wir noch erledigen, um dieses Layout aufzuräumen. Ihnen ist sicher aufgefallen, dass zwei der Zeilen höher sind als die anderen, und das liegt an den Zeilenbeschriftungen, die dank eines Umbruchs über zwei Zeilen gehen. Wenn wir diesen Umbruch verhindern können, werden alle Reihen die gleiche Höhe einnehmen. Da wir gerade dabei sind, wollen wir auch die Rahmen zwischen den Zeilenbeschriftungen abschwächen, indem wir den Rahmen ausdünnen.

```
th {border-right: 1px solid gray; border-bottom-style: dotted;
  white-space: nowrap;}
```

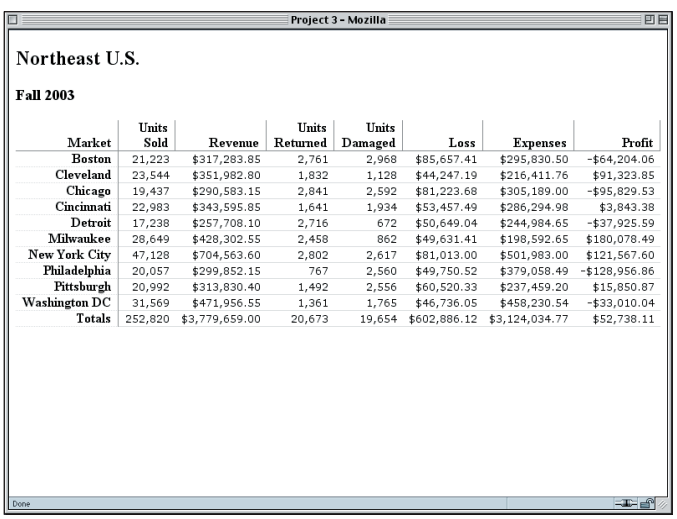
Wenn wir dies getan haben, werden die `th`-Elemente nicht umgebrochen, und das ist ein Problem, weil nun *alle* `th`-Elemente nicht umgebrochen werden – auch die Spaltenüberschriften nicht. Wenn wir das so ließen, werden einige der Spalten viel zu breit, also steuern wir diesem Effekt entgegen, indem wir die Styles der Spaltenüberschriften aktualisieren.

```
thead th {vertical-align: bottom; border: 1px solid gray;
  border-width: 0 1px 1px 0;
  white-space: normal;}
```

Das wird den `nowrap`-Wert aus der `th`-Regel außer Kraft setzen, und so kann der Text der Spaltenüberschrift nun umgebrochen werden (siehe Abbildung 3.4).

ABBILDUNG 3.4

Einheitliche Zeilenhöhe durch Verhinderung eines Umbruchs



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

3.3.4 Styles der Erträge

Nachdem wir uns um die Spalte ganz links gekümmert haben, wollen wir uns nun die ganz rechte vornehmen. Denken Sie daran, dass es eine Designvorgabe ist, die Erträge visuell hervorzuheben, auch wenn es keine besonderen Angaben darüber gibt, wie diese Hervorhebung zu bewerkstelligen ist.

Hier ist das grundlegende Problem: Uns bietet sich keine Möglichkeit, so wie das Markup nun steht, die Gewinne anders als die anderen Datenzellen zu stylen. Schauen wir uns eine Reihe von Zahlen an:

```
<tr>
<th>Cleveland</th>
<td>23,544</td>
<td>$351,982.80</td>
<td>1,832</td>
<td>1,128</td>
<td>$44,247.19</td>
<td>$216,411.76</td>
<td>$91,323.85</td>
</tr>
```

Aus Sicht des Markups sehen alle Zellen gleich aus, also müssen wir uns darum kümmern, wie wir die Zellen mit den Erträgen von den anderen unterscheiden können. Das hier sollte dem Zweck dienlich sein:

```
<td class="profit">$91,323.85</td>
```

Das muss für jede Zelle aus der Ertrag-Spalte gemacht werden, einschließlich des `th`–Elements in der Zeile der Spaltenüberschriften. Wenn diese Zellen erst einmal alle in der gleichen Klasse sind, können sie auch zusammen gestylt werden.

Das ist notwendig, weil HTML-Tabellen eigentlich keine Spalten beinhalten; sie haben nur Zeilen und Zellen. Die Spalten sind – bestenfalls – implizite Effekte, die aus der Art entstehen, wie die Zellen jeder Reihe sich an den anderen ausrichten. Weil nun alle Reihen in unserer Berichtstabelle die gleiche Anzahl von Zellen haben, hat die letzte Zelle in jeder Reihe offensichtlich mit all den anderen letzten Zellen eine Spalte gemeinsam. Aus rein struktureller Sicht gibt es allerdings keine Spalte, die gestylt werden kann.

Da wir sozusagen freies Spiel haben, wie die Zahlen der Erträge hervorgehoben werden sollen, geben wir ihnen einen grünen Hintergrund und trennen sie mit weißen Linien ab. Das wird zusammen mit den Zeilentrennern einen interessanten »Ausstanz«-Effekt schaffen, wie in Abbildung 3.5 zu sehen ist.

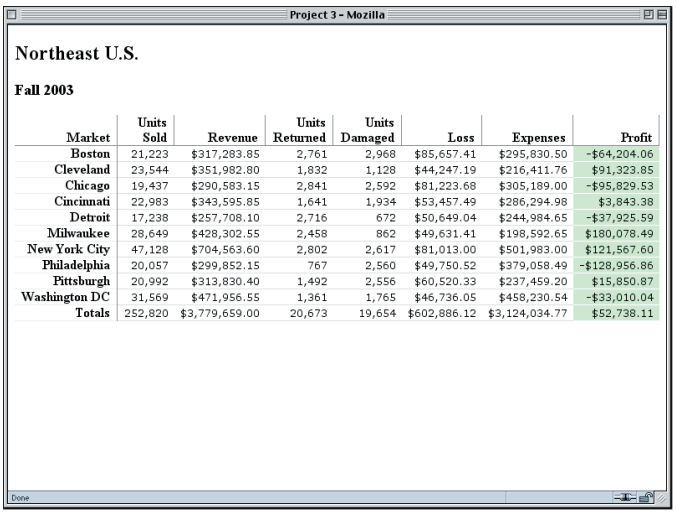
```
th {border-right: 1px solid gray; border-bottom-style: dotted;
  white-space: nowrap;}
td.profit {background: #CEC; border-bottom-color: white;}
</style>
```

HTML-Spalten

Wie einigen Lesern zweifelsohne bewusst ist, enthalten HTML und XHTML sehr wohl zu Spalten gehörige Elemente wie `col` und `colgroup`. Die haben wir geflissentlich beiseite gelassen, weil es nicht einfach ist, sie konsistent zu stylen, und weil es Uneinigkeit darüber gibt, wie Spalten überhaupt gestylt werden sollten. Es ist zwar plump, aber allgemein einfacher, Zellen eine Klasse zuzuweisen, wie wir es hier auch gemacht haben, falls spaltenorientiertes Styling gewünscht wird.

ABBILDUNG 3.5

Die Gewinne werden hervorgehoben.



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

3.3.5 Akzente für das Negative

Jetzt müssen wir die Aufmerksamkeit der Schattenseite des Business zuwenden: die negativen Ertragszahlen (auch als Verluste bekannt). Uns ist aufgetragen worden, dass sie rot auf gelbem Grund sein sollen, aber erneut stellt sich uns das Problem, wie diese Zellen mit negativen Zahlen von den anderen unterschieden werden können. Hier haben wir eine solche Zelle:

```
<td class="profit">-$64,204.06</td>
```

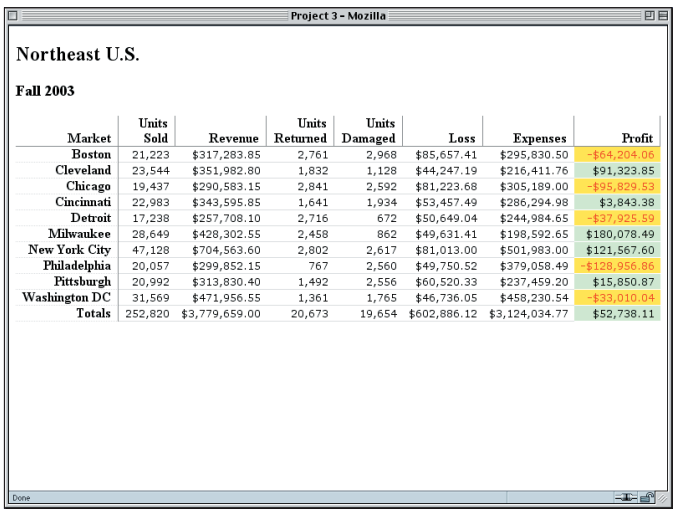
Wir haben keine andere Möglichkeit, außer selbst hinzuschauen, ob der Inhalt der Zelle einen positiven oder einen negativen Betrag aufweist. CSS kümmert sich naturbedingt nur um das Styling von Inhalten und geht dafür von der Struktur aus. Es kann den Inhalt nicht basierend auf dem Inhalt selbst stylen. Also müssen wir der Struktur selbst diese Informationen hinzufügen. Das erledigen wir, indem wir jede negative Zahl in eine `neg`-Klasse stecken.

```
<td class="profit neg">-$64,204.06</td>
```

Hier machen wir uns die Natur von `class`-Werten bei HTML zunutze, die als eine mit Leerzeichen getrennte Liste von einem oder mehreren Worten definiert sind. Besser noch, wir können auf jedes einzelne Wort unabhängig von den anderen zielen. Somit können wir eine Regel hinzufügen, die jede Zelle mit der Klasse `neg` stylt, egal ob sie noch eine weitere Klasse hat oder nicht.

```
td.profit {background: #CEC; border-bottom-color: white;}
td.neg {background: #FF3; color: red;}
</style>
```

Das wird die gewünschten Vorder- und Hintergrundfarben anwenden, ohne dass andere Styles übergangen werden. Beachten Sie, dass in Abbildung 3.6 die Zellen mit den negativen Zahlen immer noch unten einen weißen Rahmen haben. Das kommt von der Regel `td.profit`.



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

ABBILDUNG 3.6

Wir heben die Verluste hervor.

Theoretisch könnten wir Regeln schaffen, die nur auf Elemente angewendet werden, die zwei oder mehr `class`-Worte haben. Wir könnten beispielsweise alle Elemente, die als Werte sowohl die Klasse `profit` als auch `neg` haben, wie folgt stylen:

```
td.neg.profit {font-weight: bold;}
```

Der Internet Explorer kommt mit dieser Syntax nicht sonderlich gut klar, also wird das auch nicht oft angewendet. Die Styles, die wir verwendet haben, um die Abbildung 3.6 zu erstellen, werden andererseits sowohl vom Explorer als auch von allen anderen modernen Browsern unterstützt.

3.3.6 Styles der Summen

Unser letztes Ziel für das Stylen für den Bildschirm ist, dass die Summen sich durch fette Schrift unterscheiden. Dafür weisen wir der Zeile, in der die Summen stehen, eine Klasse zu.

```
<tr class="totals">
```

Jetzt ist es recht simpel, die Zelleninhalte fett ausgeben zu lassen.

```
td.neg {background: #FF3; color: red;}
tr.totals td {font-weight: bold;}
</style>
```

Tabellenfußzeilen

Wir hätten auch die letzte Reihe in ein `tfoot`-Element einschließen können, ähnlich wie die erste Reihe in ein `thead` eingeschlossen ist. In diesem Fall wählen wir `class`, um zu veranschaulichen, wie Zeilen- und Zellenklassen gemeinsam genutzt werden können.

Um der Tabelle den letzten Schliff zu verpassen, legen wir einen soliden grauen unteren Rahmen in die Zellen und verdichten den unteren Rahmen der Zeilenbeschriftung.

```
tr.totals td {font-weight: bold; border-bottom: 1px solid gray;}
tr.totals th {border-bottom-style: solid;}
</style>
```

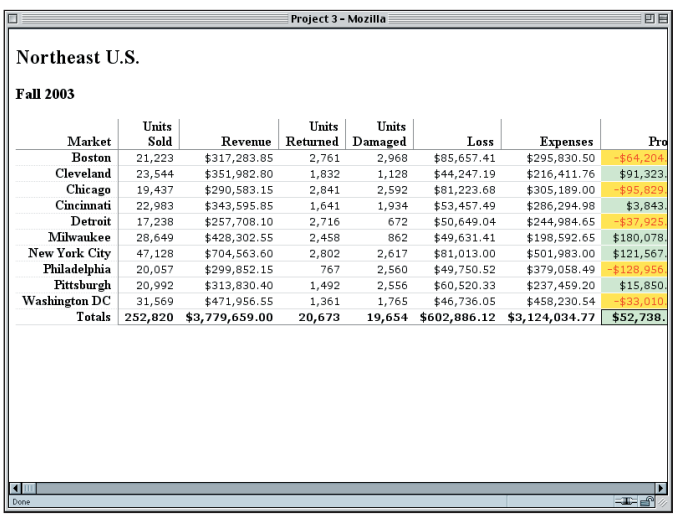
Schließlich legen wir einen Kasten um die Zahl der Gesamtsumme, weil das wohl der Teil ist, der den Leuten am wichtigsten ist.

```
tr.totals td {font-weight: bold; border-bottom: 1px solid gray;}
tr.totals td.profit {border: 1px solid black;}
tr.totals th {border-bottom-style: solid;}
```

Das Ergebnis all dieser Änderungen sehen Sie in Abbildung 3.7.

ABBILDUNG 3.7

Die Summen werden gestylt.



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Pro
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.

Dies bringt nun ein potenzielles Problem mit unserem Layout zum Vorschein: Wenn die Tabelle zu breit wird, dann wird eine Leiste für horizontales Scrollen nötig. Natürlich gilt das für Tabellen seit dem Tag, als sie durch die Netscape-Version 1.x eingeführt wurden, also ist das wahrlich nicht neu. Durch CSS haben wir Möglichkeiten, mit dieser Situation fertig zu werden. Das kann unter anderem so funktionieren:

- Sie könnten versuchen, eine explizite Breite für die `td`-Elemente festzulegen, und hoffen, dass es ausreicht, damit die Daten dort hineinpassen, aber keine Scrollbar nötig wird. Das ist wahrscheinlich auch nicht viel besser, als es der Tabelle zu überlassen, wie sie sich automatisch anordnet.
- Sie könnten die Schriftgröße für die gesamte Tabelle reduzieren. Das beinhaltet die Möglichkeit, dass der Text bis zu dem Punkt schrumpft, der ein Lesen unmöglich macht, aber mit ein wenig Vorsicht sollte es nicht schwer fallen, dass alles klein ist und trotzdem lesbar bleibt.

- Sie könnten den Platz zwischen den Zahlen reduzieren. Die sehen sowieso recht weit auseinander gezogen aus.

Diesen letzten Vorschlag wollen wir nun umsetzen. Wir führen eine neue Regel ein, die auf alle Datenzellen angewendet werden soll.

```
th {border-right: 1px solid gray; border-bottom-style: dotted;
  white-space: nowrap;}
td {letter-spacing: -1px;}
td.profit {background: #CEC; border-bottom-color: white;}
```

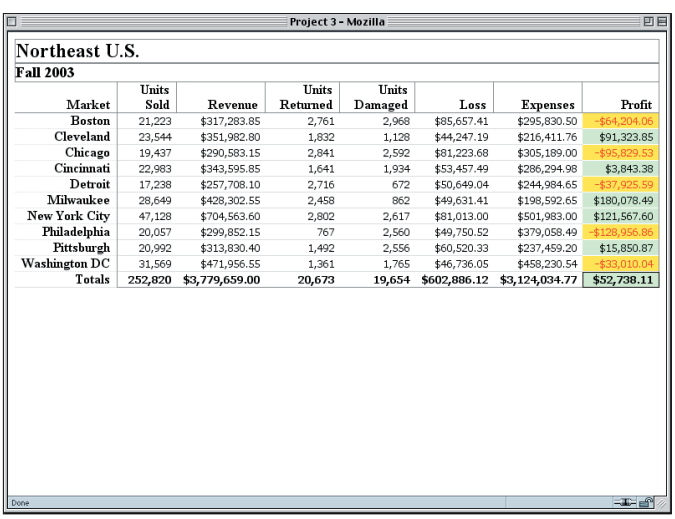
Wir könnten die gleichen Schritte für die Zeilen- und Spaltenüberschriften machen, aber damit warten wir erst einmal ab.

3.3.7 Letzte Hand anlegen

Ein paar Sachen können wir noch machen, damit der Bericht besser ausschaut. Zuerst einmal können wir einen grauen Rahmen auf die rechte Seite der Spalte Erträge legen.

```
td.profit {background: #CEC; border-bottom-color: white;
  border-right: 1px solid gray;}
```

Jetzt wollen wir uns den `h2`- und `h3`-Elementen zuwenden, die vor der eigentlichen Tabelle kommen. Es wäre interessant, wenn sich eine Linie von der rechten Seite der Tabelle aus erstreckt und ihren Weg zwischen dem Berichtstitel und seinem Datum nimmt, also machen wir das gleich mal. Wir erledigen das, indem wir beiden Headings einen Rahmen geben und deren Ränder auf Null setzen. Das Ergebnis können Sie in Abbildung 3.8 sehen.



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

ABBILDUNG 3.8
Styles auch für die Tabellenkopfzeile

```
<style type="text/css">
h2, h3 {margin: 0; border: 1px solid gray;}
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
```

Es ist klar, dass die Geschichte nicht so bleiben kann, also werden wir eine weitere Regel hinzugeben, um alle `h2`-Rahmen außer dem linken wegzunehmen, und fügen obendrein noch etwas Padding hinzu.

```
h2, h3 {margin: 0; border: 1px solid gray;}
h2 {border-width: 0 0 0 1px; padding: 0 0 0 0.25em}
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
```

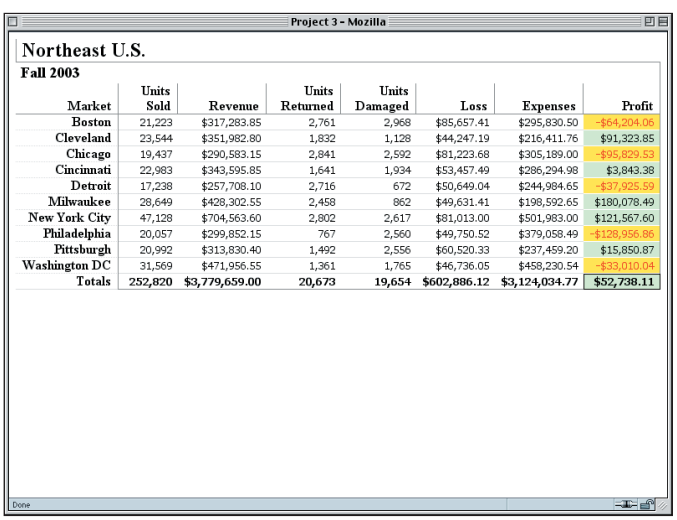
Jetzt brauchen wir dem `h3` nur noch einen oberen und rechten Rahmen und ein wenig Padding zu spendieren.

```
h2 {border-width: 0 0 0 1px; padding: 0 0 0 0.25em}
h3 {border-width: 1px 1px 0 0; padding: 0.1em 0.33em;}
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
```

Weil es keine Trennung zwischen dem `h3` und dem `table` gibt (weil keins von beiden einen Rand hat), wird der rechte Rahmen des `h3` sich mit dem rechtsbündigen Rahmen des letzten `th`-Elements in der ersten Reihe der Tabelle ausrichten. Entsprechend werden der obere Rahmen des `h3` und der linke Rahmen des `h2` zusammentreffen, um die Illusion einer kontinuierlichen Linie zu ergeben, was wir in Abbildung 3.9 sehen können.

ABBILDUNG 3.9

Die Kopfzeilen werden optisch in die Tabelle integriert.



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

Wenn dieser Einfädeleffekt unerwünscht ist, können wir einfach den Rahmen vom `h2`-Element entfernen und nur dem `h3` einen Rahmen geben. Um die Rahmen auch bei größeren Fensterbreiten zusammen zu halten, müssen wir eine kleine Regel einfügen.

```
h3 {border-width: 1px 1px 0 0; padding: 0.1em 0.33em;}
table {width: 100%;}
th, td {text-align: right; padding: 0 0.5em;
```

Mit dieser letzten Zutat ist das von uns geschaffene Stylesheet fertig und wird in Listing 3.2 präsentiert.

Listing 3.2 Das vollständige Stylesheet

```
h2, h3 {margin: 0; border: 1px solid gray;}
h2 {border-width: 0 0 0 1px; padding: 0 0 0 0.25em}
h3 {border-width: 1px 1px 0 0; padding: 0.1em 0.33em;}
table {width: 100%;}
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
td {font: small Verdana, "Andale Mono", Courier, "Courier New",
  monospace;}
thead th {vertical-align: bottom; border: 1px solid gray;
  border-width: 0 1px 1px 0;
  white-space: normal;}
th {border-right: 1px solid gray; border-bottom-style: dotted;
  white-space: nowrap;}
td {letter-spacing: -1px;}
td.profit {background: #CEC; border-bottom-color: white;
  border-right: 1px solid gray;}
td.neg {background: #FF3; color: red;}
tr.totals td {font-weight: bold; border-bottom: 1px solid gray;}
tr.totals td.profit {border: 1px solid black;}
tr.totals th {border-bottom-style: solid;}
```

3.4 Styling für Print

Nachdem wir ein Stylesheet für den Bildschirm erstellt haben, wird es Zeit, unsere Aufmerksamkeit den in den Projektzielen definierten Print-Styles zuzuwenden. Die extra für Print formulierten Vorgaben sind die folgenden:

- Beim Ausdruck soll jede zweite Zeile einen hellgrauen Hintergrund haben, und Spalten sollen durch hellgraue vertikale Linien voneinander getrennt sein.
- Alle negativen Zahlen sollen im Ausdruck kursiv erscheinen.

Diese Ziele erscheinen zusätzlich zu den anderen Projektzielen, aber denen haben wir ja schon durch das von uns geschriebene Stylesheet Genüge getan. Wir werden dafür die Styles überarbeiten, gelegentlich auch außer Kraft setzen und sie anpassen, um sie für den Ausdruck zu optimieren.

3.4.1 Der Startpunkt

Zuerst werden wir ein zweites Stylesheet einrichten. Obwohl es hier letzten Endes um Styling für Print geht, werden wir es nach dem ersten Stylesheet einfügen:

```
</style>
<style type="text/css" media="screen">

</style>
</head>
```

Wir nehmen das Attribut `media="screen"`, weil wir die Änderungen über einen Webbrowser betrachten werden. Wenn schließlich alles fertig ist, wird `screen` auf `print` geändert – und zack! haben wir ein Print-Stylesheet.

Die andere Sache, um die wir uns zu kümmern haben, ist eine kleine Veränderung des Markups, damit wir die in den Projektzielen geforderte Hervorhebung jeder zweiten Zeile hinkriegen. Zum Zeitpunkt dieses Manuskripts sind Browser nicht in der Lage, solche Dinge nur mit CSS zu erreichen, also müssen wir unserem Markup einige Klassen hinzufügen. Um so flexibel wie möglich zu sein, werden wir die Klassen `even` und `odd` verwenden. Somit wird die Tabelle dann generell die in Listing 3.3 gezeigte Grundstruktur aufweisen. (Aus Platzgründen wurden die Zellen aus den Reihen entfernt.)

Ein besserer Weg

ECSS3 definiert allerdings eine Art, um ganz einfach abwechselnd Reihen (oder andere Muster in einer Serie von Elementen) auswählen zu können, indem man die Pseudoklasse `:nth-child()` nutzt. Zum Zeitpunkt dieses Manuskripts wird es praktisch überhaupt nicht unterstützt, also sind Klassen doch der beste Weg, um solche Effekte zu erreichen.

Listing 3.3 Die grundlegende Tabellenstruktur

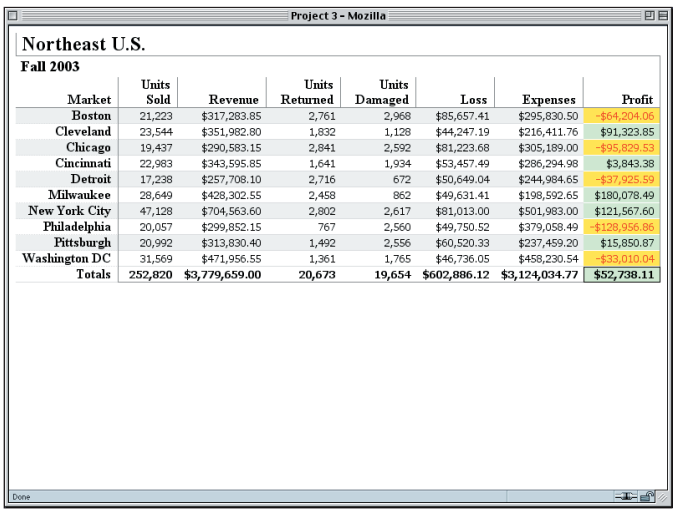
```
<table cellspacing="0">
<thead>
<tr>...</tr>
</thead>
<tbody>
<tr class="odd">...</tr>
<tr class="even">...</tr>
<tr class="odd">...</tr>
<tr class="even">...</tr>
<tr class="odd">...</tr>
<tr class="even">...</tr>
<tr class="odd">...</tr>
<tr class="even">...</tr>
<tr class="odd">...</tr>
<tr class="even">...</tr>
<tr class="totals">...</tr>
</tbody>
</table>
```

3.4.2 Hervorhebung von Reihen

Jetzt, wo wir allen Reihen eine entsprechende Klasse zugeordnet haben (Sie wissen ja, dass die erste Reihe die 1 ist, und 1 ist eine ungerade Zahl), können wir fortfahren

und jede zweite Reihe hervorheben. Wir haben uns dafür entschieden, die ungeraden Reihen hervorzuheben (siehe Abbildung 3.10).

```
<style type="text/css" media="screen">
tr.odd {background: #EEE;}
</style>
```



Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

ABBILDUNG 3.10

Über die Klassen `odd` *und* `even` *wird die jeweils nächste Reihe hervorgehoben.*

Schon ganz hübsch, aber da ist ein Problem. Die Zahlen der Erträge nehmen den Style zum Hervorheben nicht an, doch das wir wollen aber. Wir könnten den Selektor von `tr.odd` auf `tr.odd td` ändern, und damit hätten die Ertragszellen in den ungeraden Reihen dann auch eine Hervorhebung. Weil die Städtenamen sich allerdings in `th`-Elementen befinden, werden sie diesen Hervorhebungsseffekt aber verlieren. Also werden wir einen universalen Selektor nehmen, um beide auszuwählen.

```
table tr.odd * {background: #EEE;}
```

Diese kleinen, einfachen Änderungen bedeuten, dass *alle* Elemente, die innerhalb eines `tr` auftreten und eine `class` haben, in der das Wort `odd` vorkommt und die von einem `table`-Element abstammt, ausgewählt werden und somit die Hintergrundfarbe `#EEE` bekommen. Sowohl das Element `th` als auch das Element `td` stammen von den `odd`-Reihen ab, also werden auch beide direkt ausgewählt.

Das bedeutet jedenfalls, dass die Ertragszellen (über die Regel aus unserem ersten Stylesheet) in den gerade Reihen immer noch einen grünen Hintergrund bekommen. Das müssen wir außer Kraft setzen, und das Ergebnis sehen Sie in Abbildung 3.11.

```
table tr.odd * {background: #EEE;}
td.profit, td.neg {color: #000; background: #FFF;}
</style>
```

Wir haben es nun geschafft, die Vorder- und Hintergrundfarben aus den Ertrags- und Verlustzahlen zu entfernen, aber ein genauer Blick auf Abbildung 3.11 enthüllt ein wei-

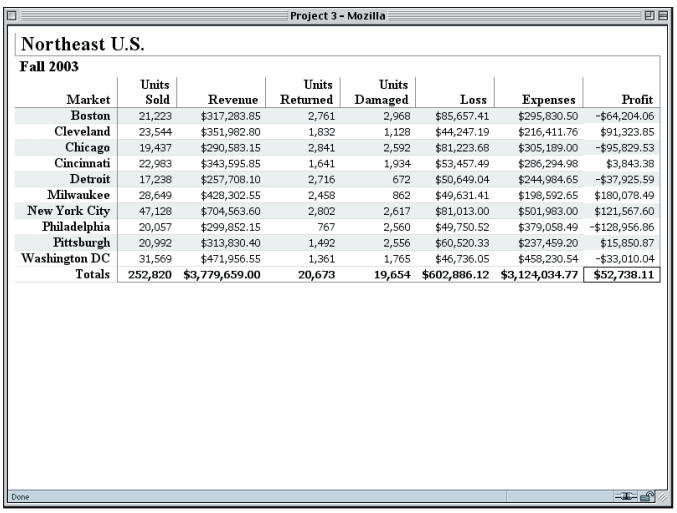
Wir werden spezifischer

Wir haben die Konstruktion `table tr.odd *` verwendet, weil wir durch das Hinzufügen von `table` sichergestellt haben, dass dieser Selektor eine größere Spezifität (Gewichtung) hat als die Selektoren `td.profit` und `td.neg`, die in dem Gesamt-Medien-Stylesheet erscheinen, das wir im ersten Teil des Projekts geschrieben haben, und die sowohl für Print als auch für Bildschirm gelten. Wenn wir `table` weggelassen hätten, hätten die Selektoren aus dem Gesamt-Medien-Stylesheet die gleiche Spezifität gehabt wie der Selektor, den wir nun für Print gerade schreiben. Wenn Selektoren die gleiche Spezifität haben (das heißt, der zuletzt Deklarierte gewinnt), würde sich die Darstellung ändern, wenn die Reihenfolge im Stylesheet umgekehrt wird. Da wir nun die Print-Deklaration spezifischer gemacht haben, gewährleisten wir, dass sie immer gewinnen wird und darum robuster ist.

teres Problem. Der untere Rahmen jeder Zelle in der Ertragsspalte ist weiß, während der untere Rahmen der normalen Zellen immer noch hellgrau ist. Das war sinnvoll, als die Ertragszellen Hintergrundfarben hatten, aber jetzt sieht es einfach nur hässlich aus.

ABBILDUNG 3.11

Ausweitung der Zeilenhervorhebung und Entfernen der farblichen Betonung



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

Druckausgabe des Hintergrunds

Auch wenn Sie eine Hintergrundfarbe für den Druck definieren, gibt es keine Garantie, dass diese auch ausgegeben wird. Die meisten Browser sind so konfiguriert, dass ein Hintergrund nicht ausgedruckt wird, meist um Tinte oder Toner zu sparen. Der User kann diese Konfiguration ändern, um die Druckausgabe der Hintergründe von Elementen zu aktivieren, aber Sie als Autor können das nicht.

Weil wir abwechselnde Zeilen hervorgehoben haben, bietet sich uns die Option, die Rahmen zwischen den Zeilen insgesamt zu beseitigen, aber stattdessen werden wir sie konsistent zu der Farbe machen, die wir zur Verdeutlichung für die jeweils zweite Zeile genommen haben.

```
table tr.odd * {background: #EEE;}
tr.odd *, tr.even * {border-bottom: 1px solid #EEE;}
td.profit, td.neg {background: #FFF;}
```

Das wird gewährleisten, dass auch wenn keine Hintergründe gedruckt werden, die Zeilen eine gewisse visuelle Trennung aufweisen. Wir sollten ebenfalls die Spalten-Separatoren hinauswerfen, die in den Designvorgaben gefordert waren.

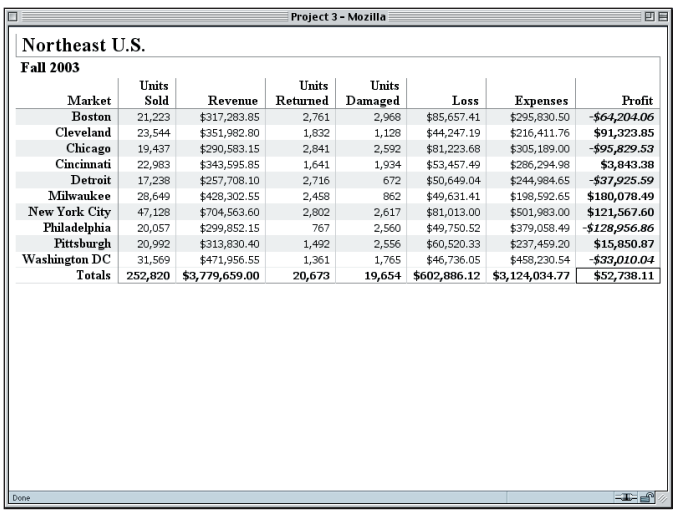
```
tr.odd *, tr.even * {border-bottom: 1px solid #EEE;}
td {border-right: 1px solid #CCC;}
td.profit, td.neg {color: #000; background: #FFF;}
```

Indem wir den Rahmen auf der rechten Seite eines jeden `td`-Elements platzieren, erhalten wir Spalten-Separatoren. In Fällen wie bei den Ertragszellen gibt es bereits einen rechten Rahmen, der durch eine Regel mit einem spezifischeren Selektor (`td.profit`) definiert wurde, also wird diese letzte Regel sie nicht ändern.

Da wir gerade von den Ertragszellen sprechen – wir müssen immer noch die Beträge der Erträge visuell akzentuieren, weil das eine der Designvorgaben ist. Wir haben die Hintergründe herausgenommen, also werden wir die Zahlen stattdessen fett ausgeben.

Den Vorgaben entsprechend haben wir auch die negativen Zahlen kursiv gemacht. Dies wird in Abbildung 3.12 gezeigt.

```
td.profit, td.neg {color: #000; background: #FFF;}
td.profit {font-weight: bold;}
td.neg {font-style: italic;}
</style>
```



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	*-$64,204.06*
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	*-$95,829.53*
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	*-$37,925.59*
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	*-$128,956.86*
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	*-$33,010.04*
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

ABBILDUNG 3.12

Sowohl die Gewinne als auch die Verluste erhalten die gebührende Aufmerksamkeit.

3.4.3 Die Summenzeile

Als Letztes werden wir die Summenzeile aufräumen. Das machen wir hauptsächlich deswegen, damit das Endergebnis besser ausschaut, nicht weil es explizit Teil der Designvorgaben ist. Durch genaues Betrachten der Abbildung 3.12 erkennen wir, dass wir schnell die Summenzeile etwas mehr hervorheben können, indem wir einen soliden oberen Rahmen hinzugeben. Das wollen wir dann auch sowohl auf die `td`- als auch die `th`-Elemente in der Zeile anwenden.

```
td.neg {font-style: italic;}
tr.totals * {border-top: 1px solid gray;}
</style>
```

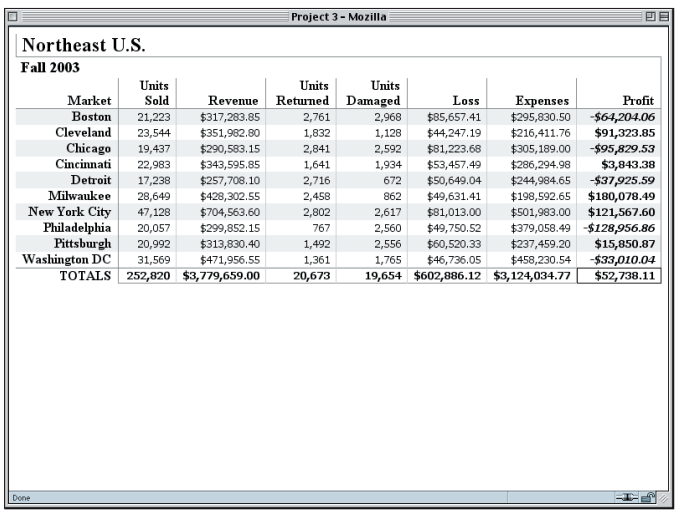
Es täte dem Feinschliff auch ganz gut, wenn die Zelle für die Gesamtsumme keinen unteren Rahmen mehr hätte, weil sie noch irgendwie dazwischen hängt. Und als I-Tüpfelchen werden wir das Wort für die Gesamtsumme in Großbuchstaben schreiben: »TOTALS«.

```
tr.totals * {border-top: 1px solid gray;}
tr.totals th {border-bottom: none; text-transform: uppercase;}
</style>
```

Dank `text-transform` können wir den Text in Großbuchstaben ausgeben, ohne die HTML-Quelle anzurühren. In diesem Fall können wir das auch verwenden, weil die fragliche Zelle tatsächlich Buchstaben enthält – bei Zahlen macht das Umwandeln in Großbuchstaben allgemein nicht viel Sinn. Das Ergebnis sehen wir in Abbildung 3.13.

ABBILDUNG 3.13

Der letzte Feinschliff für den Bericht



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	*-$64,204.06*
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	*-$95,829.53*
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	*-$37,925.59*
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	*-$128,956.86*
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	*-$33,010.04*
TOTALS	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

Somit haben wir alles fertig, damit diese Styles nur beim Print angewendet werden können. Wir brauchen dafür nur noch zum Anfang unseres zweiten Stylesheets zu gehen und – wie vorher schon beschrieben – `screen` in `print` zu ändern. Diese letzte Änderung wird in Listing 3.4 gezeigt, das beide von uns kreierten Stylesheets zeigt.

Styles für alle Medien

Das erste Stylesheet in Listing 3.4 wird auf alle Medien angewendet, weil für das Attribut `media` der Standardwert `all` lautet. Darum wird das erste Stylesheet beim Drucken mit dem zweiten kombiniert.

Listing 3.4 Beide Stylesheets zusammen

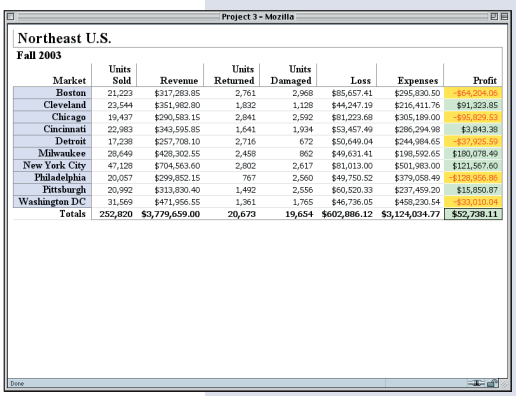
```
<style type="text/css">
h2, h3 {margin: 0; border: 1px solid gray;}
h2 {border-width: 0 0 0 1px; padding: 0 0 0 0.25em}
h3 {border-width: 1px 1px 0 0; padding: 0.1em 0.33em;}
th, td {text-align: right; padding: 0 0.5em;
  border-bottom: 1px solid #DDD;}
td {font: small Verdana, "Andale Mono", Courier, "Courier New",
  monospace;}
thead th {vertical-align: bottom; border: 1px solid gray;
  border-width: 0 1px 1px 0;
  white-space: normal;}
th {border-right: 1px solid gray; border-bottom-style: dotted;
  white-space: nowrap;}
td {letter-spacing: -1px;}
td.profit {background: #CEC; border-bottom-color: white;
  border-right: 1px solid gray;}
td.neg {background: #FF3; color: red;}
tr.totals td {font-weight: bold; border-bottom: 1px solid gray;}
```

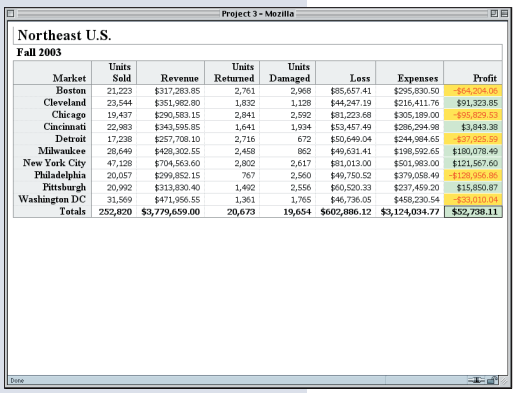
```
tr.totals td.profit {border: 1px solid black;}
tr.totals th {border-bottom-style: solid;}
</style>
<style type="text/css" media="print">
table tr.odd * {background: #EEE;}
tr.odd *, tr.even * {border-bottom: 1px solid #EEE;}
td {border-right: 1px solid #CCC;}
td.profit, td.neg {color: #000; background: #FFF;}
td.profit {font-weight: bold;}
td.neg {font-style: italic;}
tr.totals * {border-top: 1px solid gray;}
tr.totals th {border-bottom: none; text-transform: uppercase;}
</style>
```

3.5 Spielwiese

Versuchen Sie, durch das Erstellen von Styles jede der hier beschriebenen Variationen zu erreichen. Wenn das Markup geändert werden muss, damit die Variation erreicht oder überhaupt erst möglich wird, wird das im Text angegeben.

1. Geben Sie den Namen der Märkte einen visuellen Akzent, der anders ist als der bei den Ertragszahlen. Wenn Sie das Markup der Datei nehmen, die in Abbildung 3.9 gezeigt wird, ist eine Änderung durch das Hinzufügen gewisser Informationen erforderlich. Wenn Sie das Markup der Datei aus Abbildung 3.13 verwenden, braucht am Markup nichts gemacht zu werden.



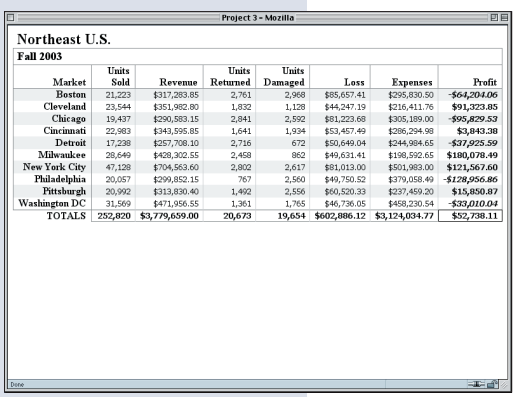


Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	-$64,204.06
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	$91,323.85
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	-$95,829.53
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	$3,843.38
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	-$37,925.59
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	$180,078.49
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	$121,567.60
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	-$128,956.86
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	$15,850.87
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	-$33,010.04
Totals	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

2. »Füllen« Sie die obere Zeile und linke Spalte auf und erstellen Sie einen Kasten um die gesamte Tabelle von Überschriften und Zahlen. Das sollten Sie auf mehrere Arten erreichen können, aber nehmen Sie die Herausforderung an, dies anders als einfach über das Setzen eines Rahmens für das `table`-Element selbst zu erreichen.



Northeast U.S.

Fall 2003

Market	Units Sold	Revenue	Units Returned	Units Damaged	Loss	Expenses	Profit
Boston	21,223	$317,283.85	2,761	2,968	$85,657.41	$295,830.50	*-$64,204.06*
Cleveland	23,544	$351,982.80	1,832	1,128	$44,247.19	$216,411.76	**$91,323.85**
Chicago	19,437	$290,583.15	2,841	2,592	$81,223.68	$305,189.00	*-$95,829.53*
Cincinnati	22,983	$343,595.85	1,641	1,934	$53,457.49	$286,294.98	**$3,843.38**
Detroit	17,238	$257,708.10	2,716	672	$50,649.04	$244,984.65	*-$37,925.59*
Milwaukee	28,649	$428,302.55	2,458	862	$49,631.41	$198,592.65	**$180,078.49**
New York City	47,128	$704,563.60	2,802	2,617	$81,013.00	$501,983.00	**$121,567.60**
Philadelphia	20,057	$299,852.15	767	2,560	$49,750.52	$379,058.49	*-$128,956.86*
Pittsburgh	20,992	$313,830.40	1,492	2,556	$60,520.33	$237,459.20	**$15,850.87**
Washington DC	31,569	$471,956.55	1,361	1,765	$46,736.05	$458,230.54	*-$33,010.04*
TOTALS	252,820	$3,779,659.00	20,673	19,654	$602,886.12	$3,124,034.77	$52,738.11

3. Überarbeiten Sie bei den Print-Styles die Rahmen, so dass die Zeilen mit den Märkten aus der Haupttabelle von einem Kasten eingefasst werden, und das soll auch für `h3` gelten, aber nicht für `h2`, das dann keine seitlichen Rahmen haben soll. Dafür müssen Sie wohl ebenfalls das Markup ändern.

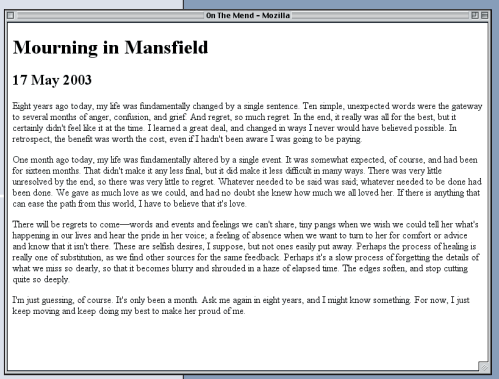
4

POSITIONIEREN IM HINTERGRUND

> *Mir haben sie aber gesagt, dass kein Mathe vorkommt! So ein Mist.*
>
> *Macintosh Technical Note # 31*

Es ist ganz üblich – wenigstens im Printdesign –, abgestufte Variationen eines Hintergrundes zu verwenden, um Teile des Designs hervorzuheben. Ein gutes Beispiel ist eine Werbeanzeige, in der das große Bild eines Berges, eines Strandes oder einer schönen Frau die gesamte Anzeige füllt und in deren Mitte ein überschwänglicher, doch bedeutungsloser Text steht, und um diesen Bereich herum gibt es eine Region, in der das Bild im Hintergrund ausgewaschen erscheint, als wäre der Text auf einem halb durchsichtigen Plastikblock geschrieben.

Da zum Zeitpunkt dieses Manuskripts Transparenz- oder Opazität-Styles noch nicht Bestandteil von CSS sind, geht man allgemein davon aus, dass diese Effekte im Prinzip unmöglich sind. Es gibt fixed-attachment-Hintergründe (schauen Sie sich das Projekt 11 in **Eric Meyer on CSS** für weitere Details an), aber der Internet Explorer für Windows unterstützt sie nicht. Man kann halbtransparente PNG-Grafiken nutzen, aber auch sie werden vom IE für Windows nicht unterstützt. Tatsächlich gibt es, wenn man nicht gerade den Browser mit proprietären Behavior-Skripts hacken will, nur einen Weg, um einen glatten



Mourning in Mansfield

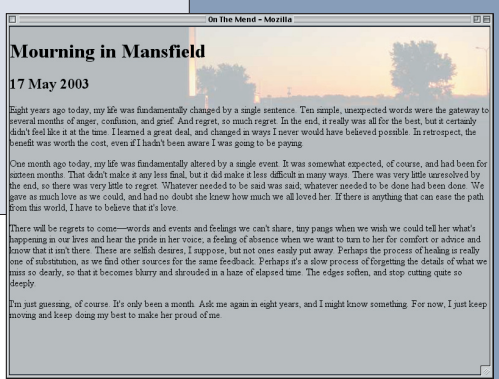
17 May 2003

Eight years ago today, my life was fundamentally changed by a single sentence. Ten simple, unexpected words were the gateway to several months of anger, confusion, and grief. And regret, so much regret. In the end, it really was all for the best, but it certainly didn't feel like it at the time. I learned a great deal, and changed in ways I never would have believed possible. In retrospect, the benefit was worth the cost, even if I hadn't been aware I was going to be paying.

One month ago today, my life was fundamentally altered by a single event. It was somewhat expected, of course, and had been for sixteen months. That didn't make it any less final, but it did make it less difficult in many ways. There was very little unresolved by the end, so there was very little to regret. Whatever needed to be said was said; whatever needed to be done had been done. We gave as much love as we could, and had no doubt she knew how much we all loved her. If there is anything that can ease the path from this world, I have to believe that it's love.

There will be regrets to come—words and events and feelings we can't share, tiny pangs when we wish we could tell her what's happening in our lives and hear the pride in her voice; a feeling of absence when we want to turn to her for comfort or advice and know that it isn't there. These are selfish desires, I suppose, but not ones easily put away. Perhaps the process of healing is really one of substitution, as we find other sources for the same feedback. Perhaps it's a slow process of forgetting the details of what we miss so dearly, so that it becomes blurry and shrouded in a haze of elapsed time. The edges soften, and stop cutting quite so deeply.

I'm just guessing, of course. It's only been a month. Ask me again in eight years, and I might know something. For now, I just keep moving and keep doing my best to make her proud of me.



Mourning in Mansfield

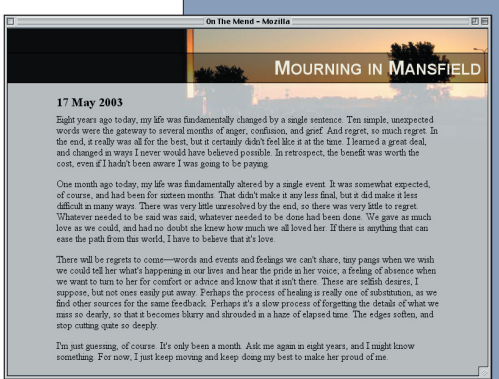
17 May 2003

Eight years ago today, my life was fundamentally changed by a single sentence. Ten simple, unexpected words were the gateway to several months of anger, confusion, and grief. And regret, so much regret. In the end, it really was all for the best, but it certainly didn't feel like it at the time. I learned a great deal, and changed in ways I never would have believed possible. In retrospect, the benefit was worth the cost, even if I hadn't been aware I was going to be paying.

One month ago today, my life was fundamentally altered by a single event. It was somewhat expected, of course, and had been for sixteen months. That didn't make it any less final, but it did make it less difficult in many ways. There was very little unresolved by the end, so there was very little to regret. Whatever needed to be said was said; whatever needed to be done had been done. We gave as much love as we could, and had no doubt she knew how much we all loved her. If there is anything that can ease the path from this world, I have to believe that it's love.

There will be regrets to come—words and events and feelings we can't share, tiny pangs when we wish we could tell her what's happening in our lives and hear the pride in her voice; a feeling of absence when we want to turn to her for comfort or advice and know that it isn't there. These are selfish desires, I suppose, but not ones easily put away. Perhaps the process of healing is really one of substitution, as we find other sources for the same feedback. Perhaps it's a slow process of forgetting the details of what we miss so dearly, so that it becomes blurry and shrouded in a haze of elapsed time. The edges soften, and stop cutting quite so deeply.

I'm just guessing, of course. It's only been a month. Ask me again in eight years, and I might know something. For now, I just keep moving and keep doing my best to make her proud of me.



MOURNING IN MANSFIELD

17 May 2003

Eight years ago today, my life was fundamentally changed by a single sentence. Ten simple, unexpected words were the gateway to several months of anger, confusion, and grief. And regret, so much regret. In the end, it really was all for the best, but it certainly didn't feel like it at the time. I learned a great deal, and changed in ways I never would have believed possible. In retrospect, the benefit was worth the cost, even if I hadn't been aware I was going to be paying.

One month ago today, my life was fundamentally altered by a single event. It was somewhat expected, of course, and had been for sixteen months. That didn't make it any less final, but it did make it less difficult in many ways. There was very little unresolved by the end, so there was very little to regret. Whatever needed to be said was said; whatever needed to be done had been done. We gave as much love as we could, and had no doubt she knew how much we all loved her. If there is anything that can ease the path from this world, I have to believe that it's love.

There will be regrets to come—words and events and feelings we can't share, tiny pangs when we wish we could tell her what's happening in our lives and hear the pride in her voice; a feeling of absence when we want to turn to her for comfort or advice and know that it isn't there. These are selfish desires, I suppose, but not ones easily put away. Perhaps the process of healing is really one of substitution, as we find other sources for the same feedback. Perhaps it's a slow process of forgetting the details of what we miss so dearly, so that it becomes blurry and shrouded in a haze of elapsed time. The edges soften, and stop cutting quite so deeply.

I'm just guessing, of course. It's only been a month. Ask me again in eight years, and I might know something. For now, I just keep moving and keep doing my best to make her proud of me.

lichtdurchlässigen Effekt beim Explorer für Windows zu bekommen, und das geht über die Manipulation der Position der Hintergrundbilder.

4.1 Projektziele

Bei diesem Projekt geht es darum, dass ein Autor einige seiner Essays veröffentlichen will und sie künstlerisch gestaltet sehen möchte. Er ist ein großer Fan von Transparenzeffekten, also möchte er solche auch in seinen Designs sehen. Genauer gesagt:

- Für sein erstes Essay »Mourning in Mansfield« sollen wir das Bild eines Sonnenaufgangs nehmen. Darin soll der ganze Hintergrund hinter dem Titel schattig abgedunkelt sein, und für den Haupttext des Essays soll als Effekt der Hintergrund aufgehellt werden.
- Für den Essay »Gathering Stormclouds« werden wir das Bild einiger Wolken bei Sonnenuntergang nehmen. Bei diesem Design wird als Effekt der Titel vor dem Hintergrund aufgehellt, während der Haupttext einen abgedunkelten Hintergrund und einen hellen Text haben wird.

Der Autor stellt uns die Bilder zur Verfügung, also brauchen wir uns darum zum Glück nicht kümmern. Wir brauchen da eigentlich nur einen kleinen Styling-Trick durchzuführen, um die von unserem Klienten gewünschten Transparenzeffekte zu bekommen.

4.2 Vorbereitungen

Laden Sie die Dateien für Projekt 4 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch04proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und erneut laden.

4.3 Styles der Abenddämmerung

Für die erste Hälfte dieses Projekts werden wir das erste der beiden Dokumente nehmen und die nötigen Styles hinzufügen, um Transparenzeffekte mit einfachen JPEG-Bildern zu kreieren. Wie wir bald sehen werden, machen es uns die besagten Bilder sehr leicht, ein attraktives Design zu gestalten.

4.3.1 Der Anfang

Wie gewöhnlich sollte unser erster Schritt darin bestehen, dass wir uns die Struktur des Dokuments anschauen. Wie wir dem Listing 4.1 entnehmen können, ist kaum etwas vorhanden, nur ein »Masthead«-`div` mit der Überschrift und ein »Main«-`div` mit dem eigentlichen Text. Ohne irgendwelche Styles bietet sich uns die ganz schlichte Darstellung aus Abbildung 4.1.

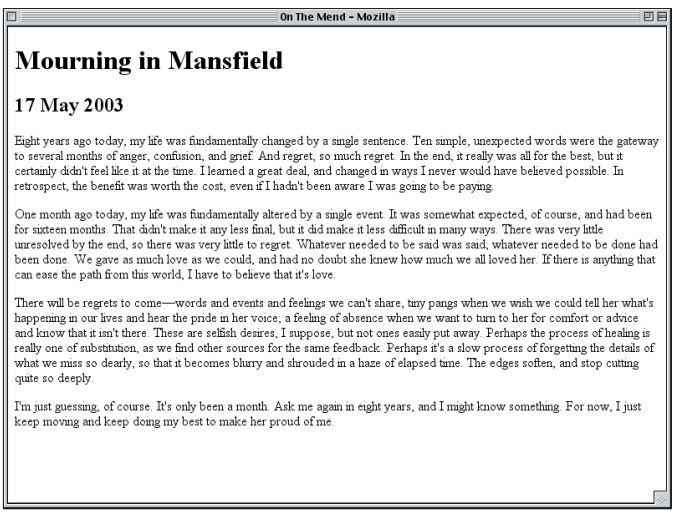


Abbildung 4.1

Das Journal im ungestylten Zustand

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

Listing 4.1 Die grundlegende Dokumentstruktur

```
<div id="masthead">
  <h1>Mourning in Mansfield</h1>
</div>
<div id="main">
  <h2>17 May 2003</h2>
  [...entry text...]
</div>
```

Jetzt müssen wir mit nicht mehr als dem, was bereits im Dokument vorhanden ist, und ein paar Hintergrundbildern ein geschmackvolles Design gestalten, das Transparenzeffekte nutzt. Um dieses Ziel zu erreichen, haben wir die drei Bilder aus Abbildung 4.2: das Ausgangsbild (`morn-base.jpg`), eine abgedunkelte Version des gleichen Bildes (`morn-fade.jpg`) und eine verblasste Version (`morn-wash.jpg`).

ABBILDUNG 4.2
Die drei uns zur Verfügung stehenden Hintergrundbilder

Um die »Fugen« zu entfernen, werden wir als Erstes das `body`-Element selbst aufräumen. Das machen wir mit einer vertrauten Regel:

```
<style type="text/css">
body {margin: 0; padding: 0;}
</style>
```

Als Nächstes werden wir ein Bild in den Hintergrund des `body` legen. Es ist wichtig, hier das richtige zu wählen, weil die Hintergrundfarbe zum Hintergrundbild passen muss, und der Body-Hintergrund ist der, der vom Design am meisten zu sehen sein wird. Für bestmögliche Lesbarkeit werden wir die verblasste Version des Bildes für den Body nehmen. Eine schnelle Prüfung mit einem Tool zur Farbanalyse ergibt, dass die Hintergrundfarbe ein Grauwert mit 71 % Weiß darin ist, also machen wir Folgendes:

```
body {margin: 0; padding: 0;
  background: rgb(71%,71%,71%) url(morn-wash.jpg);}
```

Das ist schon mal was, aber wir brauchen noch mehr. So wie die Regel jetzt steht, wird das Bild in der oberen linken Ecke des Hintergrunds des Body anfangen und sowohl horizontal als auch vertikal endlos als Kachel angeordnet. Für unser Design wollen wir, dass das Bild einmal erscheint und sich *nicht* wiederholt, und wir wollen es in der oberen rechten Ecke platziert haben. Dazu können wir die Hintergrund-Deklaration noch etwas erweitern, und das führt zu dem Ergebnis in Abbildung 4.3.

```
body {margin: 0; padding: 0;
  background: rgb(71%,71%,71%) url(morn-wash.jpg) 100% 0 no-repeat;}
```

Worte statt Zahlen

Wir hätten auch die Schlüsselwörter `right top` statt der Werte `100% 0` für die Hintergrundposition nehmen können, aber weil spätere Hintergründe über Offsets positioniert werden, vermeiden wir die Verwendung von Schlüsselwörtern in diesem Projekt ganz.

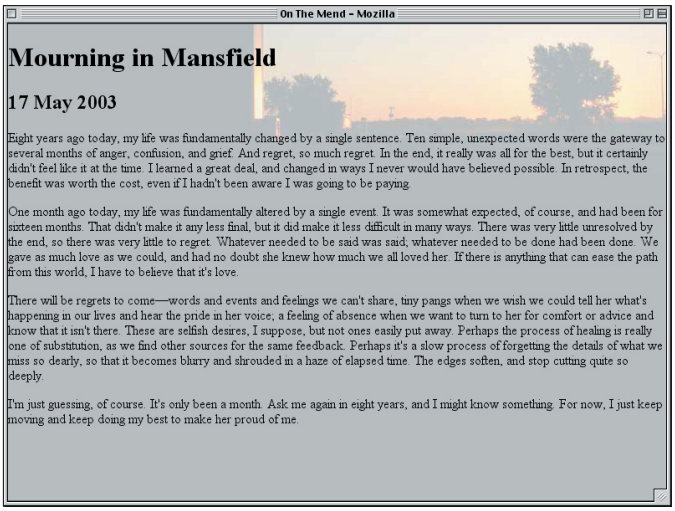


Abbildung 4.3

Ein paar grundlegende Styles im `body`-Element machen den Anfang.

4.3.2 Masthead-Styles

Den Masthead des Dokuments werden wir uns als Nächstes genauer vornehmen. Wir werden das Ausgangsbild für den Hintergrund (`morn-base.jpg`) auf den Masthead selbst anwenden, aber das »Wie« wollen wir uns mal einen Moment genauer überlegen.

Unser Ziel ist, dass die Hintergrundbilder aneinander ausgerichtet sind, um die Illusion der Transparenz zu erzeugen. Damit das geschieht, müssen wir sichergehen, dass die obere Kante des Masthead an der oberen Kante des Body ausgerichtet ist. Wir müssen auch dafür sorgen, dass das Hintergrundbild sich nicht wiederholt und genau an der gleichen Stelle sitzt wie beim Body. Darum kriegen die Werte `100% 0` und `no-repeat` noch einen Auftritt. Schnell wieder eine Farbanalyse, und wir haben den Wert für die Hintergrundfarbe, die wir passend zum Bild machen werden.

```
body {margin: 0; padding: 0;
  background: rgb(71%,71%,71%) url(morn-wash.jpg) 100% 0 no-repeat;}
#masthead {background: rgb(2%,4%,4%) url(morn-base.jpg)
 100% 0 no-repeat;}
</style>
```

Eine Sache ist da noch zu erledigen: Wir müssen vom `h1`-Element innerhalb des Masthead-`div`s die Ränder entfernen. Machen wir das nicht, werden in CSS-konformen Browsern die Ränder aus dem `div` herausragen, und das führt dazu, dass die obere Kante des `div` vom oberen Rand des Dokuments nach unten geschoben wird. Wenn wir die Ränder entfernen, entsteht das Ergebnis in Abbildung 4.4.

```
#masthead {background: rgb(2%,4%,4%) url(morn-base.jpg)
  100% 0 no-repeat;}
#masthead h1 {margin: 0;}
</style>
```

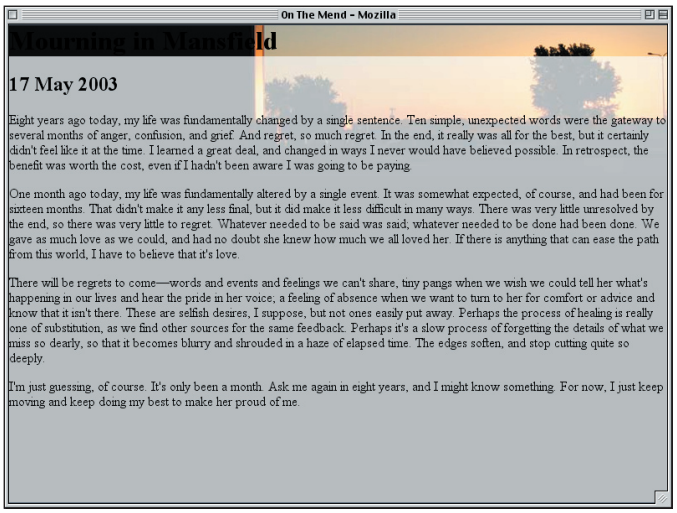


ABBILDUNG 4.4

Der Masthead bekommt ein anderes Hintergrundbild.

Da dankenswerterweise der Masthead und der Body an der oberen Kante ausgerichtet sind, sind ihre Hintergründe das auch, und das ist genau die Art Effekt, die wir gewollt haben. Obwohl der Hintergrund des Masthead eigentlich »über« dem des Bodys ist, wirkt es, als wäre da eine grauschattierte transparente Schicht zwischen uns und dem »Seitenhintergrund«. Das ist nur eine Illusion, aber eine sehr nützliche!

Natürlich können wir die Sache nicht so lassen. Der Text der Überschrift ist ja schwarz auf schwarzem Hintergrund, also nicht sonderlich lesbar. Wir wollen seine Farbe ändern, damit sie dem Thema »Sonnenaufgang« entspricht, und damit das Font-Styling ein wenig professioneller ausschaut, peppen wir gleich noch etwas auf.

```
#masthead h1 {margin: 0;
  color: #EED;
  font: small-caps bold 2em/1em Arial, sans-serif;}
</style>
```

Die Werte für `font-size` und `line-height` haben wir mit besonderem Bedacht gewählt. Indem wir den `font-size` des `h1` auf `2em` setzen, machen wir den Text doppelt so groß wie sein Elternelement, das `#masthead div`, das wiederum seinen Wert für `font-size` von seinem Elternelemente, dem `body`, geerbt hat. Mit einem Wert für `line-height` von `1em` haben wir die Höhe des Inhalts von `h1` so definiert, dass sie mit ihrem `font-size` genau gleich ist.

Nun wäre es eigentlich ganz schön, nachdem wir den Masthead-Text aufgeräumt haben, die Höhe des Mastheads aufzustocken, indem wir dem Masthead-`div` noch ein bisschen Padding geben.

```
#masthead {padding: 2.5em 0 0;
  background: rgb(2%,4%,4%) url(morn-base.jpg) 100% 0 no-repeat;}
```

Standards der Zeilenhöhen

Warum nehmen wir das so genau mit dem Wert `1em` für `line-height`? Weil das nicht der Standard ist – die meisten Browser nehmen für `line-height` einen Standardwert um die 1,2. Das macht die Höhe jeder Zeile ein wenig größer als der Wert für `font-size`. Unser Wert für `1em` überschreibt dieses Standardverhalten, und das wird sich später noch als ganz praktisch herausstellen.

Dadurch zwängen wir im Grunde einen Raum zwischen den oberen Rahmen des Masthead-`div`s und den oberen Kantenrand des `h1`. Wenn wir den Wert für das obere Padding auf `2.5em` setzen, lassen wir zu, dass der Zeilenabstand bei Änderungen der Textgröße in Proportion bleibt, egal wie diese Änderungen passieren.

Wo das nun alles erledigt ist, können wir uns die Verbesserungen in Abbildung 4.5 anschauen, obwohl noch nicht alles in trockenen Tüchern ist.

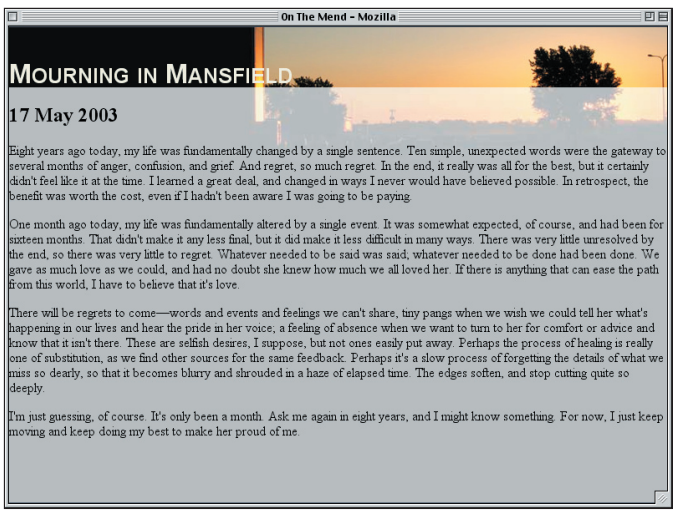


ABBILDUNG 4.5

Padding für den Masthead und Styles für die Überschriften bringen drastische Verbesserungen.

Der helle Text liegt nun auf einem Hintergrund, in dem sich hell und dunkel mischen, und das macht den Text schwer lesbar, egal welche Farbe wir da zuweisen. Wir könnten den Text auf die rechte Seite schieben, damit er mehr über dem hellen Bereich des Hintergrundes liegt, aber der dunkle Baum verhindert auch in dem Fall, dass wir dem Text eine dunkle Farbe geben.

Wir brauchen nun für die Überschrift selbst einen Hintergrund, der konsistenter ist, und glücklicherweise haben wir schon einen: `morn-fade.jpg`. Den können wir dem Hintergrund des `h1`-Elements zufügen.

```
#masthead h1 {margin: 0;
  background: rgb(4%,4%,4%) url(morn-fade.jpg) 100% 0 no-repeat;
  color: #EED;
  font: small-caps bold 2em/1em Arial, sans-serif;}
```

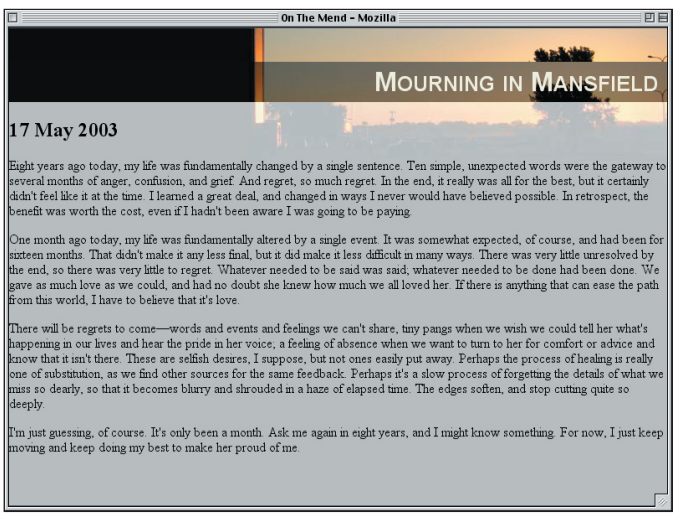
Weil wir nun für unseren hellen Text einen dunkleren Hintergrund haben, können wir ihn endlich auf die rechte Seite des Designs schieben. Zugleich werden wir das `h1`-Element ein wenig aufpolstern, damit der Text nicht zu dicht an den Rand des dunklen Bereichs heranrutscht.

```
#masthead h1 {margin: 0; padding: 0.25em 0.33em;
  background: rgb(4%,4%,4%) url(morn-fade.jpg) 100% 0 no-repeat;
  color: #EED;
  font: small-caps bold 2em/1em Arial, sans-serif;
  text-align: right;}
```

Das Padding oben und unten mit einem Viertel em wird den Text schön vor dem dunklen Hintergrund zentrieren, und das Padding mit einem Drittel em links und rechts wird verhindern, dass der Titeltext es sich mit der Kante des Browserfensters zu gemütlich macht (siehe Abbildung 4.6).

ABBILDUNG 4.6

Der dritte Hintergrund wird eingefügt, obwohl er nicht ausgerichtet ist.



Huch – da haben wir ein Problem. Das gerade eingefügte Hintergrundbild ist nicht an den anderen Bildern ausgerichtet, und darum ist der Baum unterbrochen. Das liegt daran, dass der obere Rand des Hintergrundbereichs von `h1` nicht an den oberen Kanten der anderen beiden ausgerichtet ist.

Wir könnten das geradebiegen, indem wir den `h1` auf der Seite ganz nach oben verschieben, aber das sähe nicht gerade schön aus. Stattdessen werden wir den Hintergrund von `h1` in eine Ausrichtung mit den anderen bringen, aber das Element dort belassen, wo es jetzt sitzt. Dafür brauchen wir nun ein wenig Gehirnschmalz.

Wir wissen, dass die Oberkante des `h1` 2,5 em vom Kopf des Mastheads entfernt ist, dank des oberen Paddings, das wir dem Masthead-`div` gegeben haben. Allerdings ist der `font-size` des `h1` doppelt so groß wie der des Masthead-`div`s, was an dem Wert `2em` in der `font`-Deklaration liegt, also müssen wir für den richtigen Offset diese Zahl durch zwei teilen. Wenn wir also das Hintergrundbild des `h1` so positionieren, dass seine Oberkante tatsächlich -1,25 em über der Oberkante des Hintergrundbereichs von `h1` liegt, dann sollten alle Hintergründe aneinander ausgerichtet sein.

Bemessung der ems

Sie wissen, dass ein em immer gleich dem `font-size` des Elements ist, auf das das em angewendet wird. Somit gilt bei unseren Styles, dass `1em` für das `h1` doppelt so groß ist wie `1em` für einen Absatz.

```
#masthead h1 {margin: 0; padding: 0.25em 0.33em;
  background: rgb(4%,4%,4%) url(morn-fade.jpg) 100% -1.25em no-repeat;
  color: #EED;
  font: small-caps bold 2em/1em Arial, sans-serif;
  text-align: right;}
```

Um den Look des Masthead noch etwas aufzupolieren, geben wir ihm einen soliden schwarzen Rahmen oben und unten am dunklen Hintergrund. Allerdings heißt das, dass (dank des oberen Rahmens) der Hintergrundbereich des `h1` um ein Pixel nach unten verschoben wird, und das führt in CSS-konformen Browsern dazu, dass die Bilder aus der Fluchtlinie verschoben werden. Das können wir aber mit einem negativen oberen Rand von einem Pixel ausgleichen.

```
#masthead h1 {margin: -1px 0 0; padding: 0.25em 0.33em;
  background: rgb(4%,4%,4%) url(morn-fade.jpg) 100% -1.25em no-repeat;
  color: #EED; border: 1px solid black; border-width: 1px 0;
  font: small-caps bold 2em/1em Arial, sans-serif;
  text-align: right;}
```

Ein Problem gibt es aber noch: Der Internet Explorer für Windows dreht komplett durch, wenn er auf einen negativen Rand trifft. Also werden wir nun den Rand wieder auf den vorigen Wert zurücknehmen und einen der andern Bugs vom Internet Explorer ausnutzen, um den negativen Rand vor ihm zu verstecken.

```
#masthead h1 {margin: 0; padding: 0.25em 0.33em;
  background: rgb(4%,4%,4%) url(morn-fade.jpg) 100% -1.25em no-repeat;
  color: #EED; border: 1px solid black; border-width: 1px 0;
  font: small-caps bold 2em/1em Arial, sans-serif;
  text-align: right;}
html>body #masthead h1 {margin: -1px 0 0;}
</style>
```

Dank des `html>body`-Teils dieser Regel wird der Explorer die neue Regel einfach überspringen. Andere Browser wie Safari oder Opera und Gecko-basierte Browser wie Mozilla werden diese Regel verstehen und anwenden. Also haben wir damit ein gutes browserübergreifendes Layout bekommen.

Zwar können diese neuen Styles in einigen Browsern um einen Pixel verschoben sein, aber dieser Preis ist verschmerzbar. Der Unterschied wird so oder so kaum wahrnehmbar sein, wie in Abbildung 4.7 zu sehen ist.

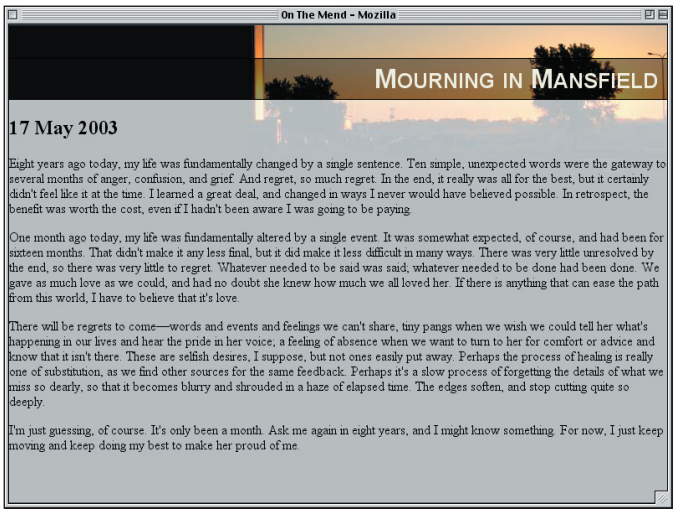


Abbildung 4.7

Mit nur ein wenig negativer Positionierung können die Hintergründe prima ausgerichtet werden.

4.3.3 Aufräumen

An diesem Punkt ist der Masthead schon gut gelungen, wir brauchen nur noch den Eintrag selbst aufzuräumen. Weil der Text mit unserem Projekt über das Ausrichten des Hintergrunds wenig zu tun hat, werden wir alle Styles auf einmal einfügen. Wenn wir dem Haupt-`div` ein paar Ränder zufügen, dem `h2` einen `font-size` und gar keine Ränder spendieren und noch etwas Rand in die Absätze kneten, sind wir fertig:

```
html>body #masthead h1 {margin: -1px 0 0;}
#main {margin: 1.25em 5em 0 5em;}
#main h2 {margin: 0; font-size: 1.25em;}
#main p {margin: 0.5em 0 1em;}
</style>
```

Damit kommen wir schließlich bei den vollständigen Styles an, die Sie in Listing 4.2 und Abbildung 4.8 bewundern können.

Listing 4.2 Das vollständige Stylesheet

```
body {margin: 0; padding: 0;
  background: rgb(71%,71%,71%) url(morn-wash.jpg) 100% 0 no-repeat;}
#masthead {padding: 2.5em 0 0;
  background: rgb(2%,4%,4%) url(morn-base.jpg) 100% 0 no-repeat;}
#masthead h1 {margin: 0; padding: 0.25em 0.33em;
  background: rgb(4%,4%,4%) url(morn-fade.jpg) 100% -1.25em no-repeat;
  color: #EED; border: 1px solid black; border-width: 1px 0;
  font: small-caps bold 2em/1em Arial, sans-serif;
  text-align: right;}
html>body #masthead h1 {margin: -1px 0 0;}
#main {margin: 1.25em 5em 0 5em;}
```

```
#main h2 {margin: 0; font-size: 1.25em;}
#main p {margin: 0.5em 0 1em;}
#masthead h1 is missing the margin declaration from above. Should read:
#masthead h1 {margin: -1px 0 0; (and so forth) - DS
```

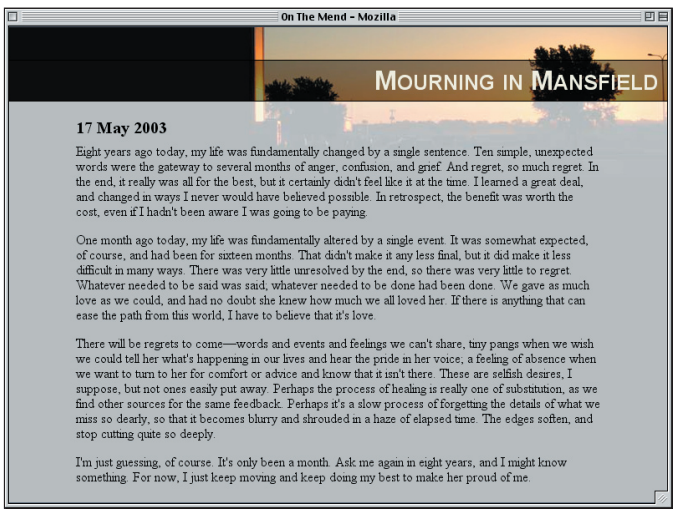


ABBILDUNG 4.8

Die vollständigen Styles des Journals

Bis zu einem gewissen Grad war unser Job in diesem Teil des Projekts recht einfach, weil zwei der drei Elemente mit Hintergrund sich an der oberen Kante ausrichten ließen. Weil das der Fall war, konnten wir zweien der Hintergrundbilder genau die gleiche Positionswerte zuweisen, und das dritte brauchte dann nur noch einen vertikalen Versatz.

Nehmen wir einmal an, dass wir wieder einen ähnlichen Transparenzeffekt erzielen wollen, aber keins der Elemente an der oberen Kante zueinander ausgerichtet ist. In so einer Situation müssen wir etwas mehr Mathematik bemühen und sorgsam mit den von uns verwendeten Längeneinheiten umgehen. Wir wollen uns nun dem zweiten Dokument in unserem Projekt zuwenden und schauen, wie solche Probleme zu bewältigen sind.

4.4 Wolkige Styles

In dieser Hälfte des Projekts werden wir uns ein neues Dokument mit einer etwas anderen Struktur vornehmen und es so überarbeiten, dass es einen transparenten Effekt bekommt. Wir werden die Prinzipien nutzen, die wir in der ersten Hälfte des Projekts erforscht haben, aber sie auf verfeinerte Art einsetzen, weil sich in diesem Dokument keins der Elemente an der oberen Kante ausrichtet.

4.4.1 Einschätzung von Struktur und Style

Wie immer besteht unser erster Schritt darin, sozusagen die Landschaft auszukundschaften, indem wir die Dokumentstruktur und alle Styles untersuchen, die eventuell schon vorhanden sind. Die Grundstruktur ist folgende:

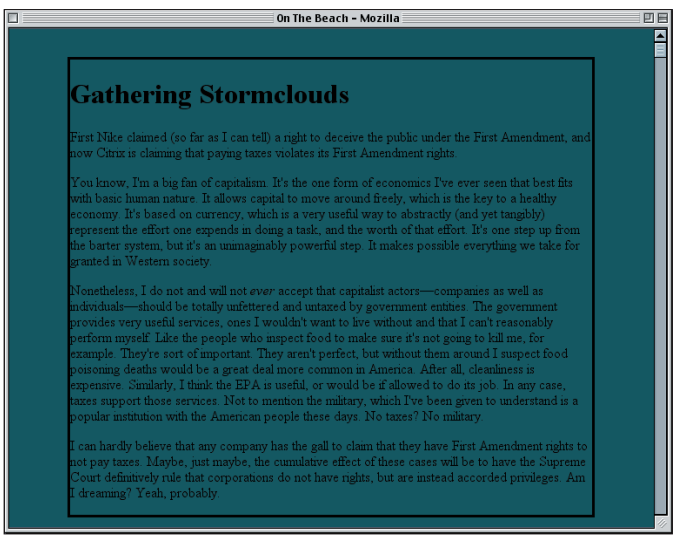
```
<div id="main">
  <h1>Gathering Stormclouds</h1>
  <div id="content">
    [...content...]
  </div>
</div>
```

Hier finden wir nun ein bereits in das Dokument eingebettetes erstes Stylesheet, und das führt zu dem in Abbildung 4.9 gezeigten Ergebnis.

```
<style type="text/css">
body {margin: 0; padding: 0;
  background: rgb(14%,26%,30%);}
#main {width: 600px; margin: 2em auto;
  border: 3px solid black;}
</style>
```

ABBILDUNG 4.9

Ein grundlegender, wenn auch düsterer Anfang für unser neues Dokument



Auto-Margins

Indem wir dem linken und rechten Rand des Haupt-`divs` den Wert `auto` und der Breite einen expliziten Wert geben, haben wir das `div` in seinem Elternelement zentriert. Beachten Sie, dass dies nicht in Versionen des Internet Explorers für Windows vor Version 6 funktioniert, der mit `text-align:center` die Elemente unkorrekt zentriert hat.

Natürlich werden wir da andere Hintergründe einsetzen, also wird der Text auch besser lesbar sein, wenn wir fertig sind. Diese Styles haben wir nur für den Anfang so eingerichtet, damit wir uns gleich in das Positionieren mit dem Hintergrund stürzen können.

Jetzt brauchen wir alle der drei in Abbildung 4.10 gezeigten Bilder: das Ausgangsbild (`storm-base.jpg`), eine abgedunkelte Version des gleichen Bildes (`storm-fade.jpg`) und eine verblasste Version (`storm-wash.jpg`).

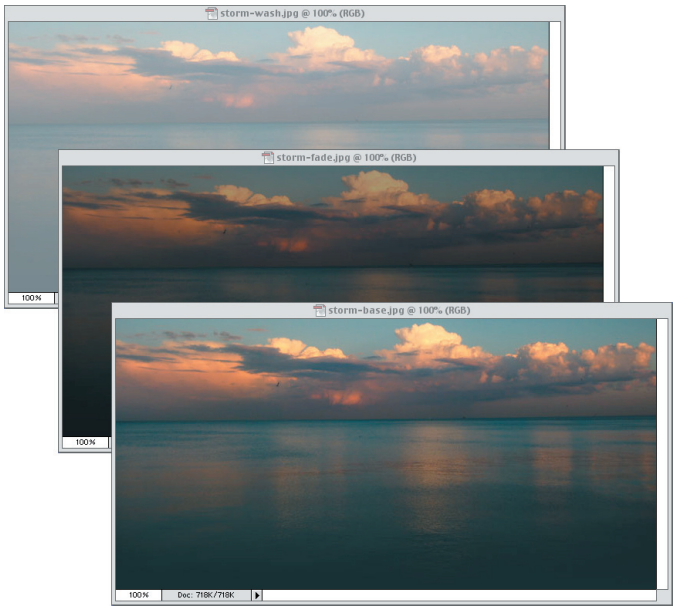


Abbildung 4.10
Die uns zur Verfügung stehenden Hintergrundbilder

Bei diesem Design werden wir keines der Bilder auf das `body`-Element anwenden. Stattdessen werden wir das Haupt-`div` und einige seiner Nachkömmlinge nehmen, aber zuerst wollen wir den Text stylen und die Elemente platzieren.

4.4.2 Titel-Styles

Beim Gestalten des Texts und der Element-Styles sollten wir als Erstes dem Haupt-`div` eine temporäre Hintergrundfarbe zuweisen. So können wir leichter sehen, was wir machen.

```
#main {width: 600px; margin: 2em auto;
  border: 3px solid black;
  background: gray;}  /* temporary background */
```

Jetzt geht es direkt ans Design. Dem Titel der Seite weisen wir Styles für Größe und Höhe zu, die wir uns aus der ersten Hälfte des Projekts kopieren.

```
#main {width: 600px; margin: 2em auto;
  border: 3px solid black;
  background: gray;}
#main h1 {font: 2em/1em "Times New Roman", serif;}
</style>
```

Das wird wie vorher den Text von `h1` in Relation zu seinem Elternelement (in diesem Fall `div#main`) setzen und gewährleisten, dass dessen `line-height` genau so groß ist wie seine `font-size`. Wir werden den Text auch zentrieren, ihn in Kleinbuchstaben setzen und gleich noch die Laufweite erhöhen – siehe Abbildung 4.11.

Fonts in Anführungsstrichen

Es ist nur dann nötig, die Namen der Fonts in Anführungsstriche zu setzen, wenn sie Zwischenräume enthalten oder Zeichen, die keine Buchstaben sind.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;}
```

ABBILDUNG 4.11

Ein temporärer Hintergrund hilft uns, die Änderungen am Seitentitel besser zu sehen.

Dass sich der Titel so zwischen die Kanten der Box quetscht, fühlt sich ein bisschen klaustrophobisch an, also schaffen wir über Ränder für `h1` ein wenig Luft.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;
  margin: 1.25em 1em 0;}
```

Um dem Bereich des Designs mit dem Text eine Definition zu spendieren, sieht es doch wohl cool aus, wenn Titel und Inhalt ein paar dünne Ränder kriegen. Wir fangen beim Titel an, aber die Ränder sollen nur oben und auf den beiden Seiten erscheinen.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;
  margin: 1.25em 1em 0;
  border: 1px solid black; border-bottom: none;}
```

Natürlich läuft das jetzt darauf hinaus, dass der Titeltext wieder dicht an einem Rahmen hängt, aber da hilft uns etwas Padding weiter.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;
  margin: 1.25em 1em 0; padding: 0.5em 0.25em;
  border: 1px solid black; border-bottom: none;}
```

Ihnen ist wahrscheinlich schon aufgefallen, dass wir bei den Werten für Ränder und Padding darauf achten, dass sie durch 0,25 teilbar sind. Das wird es uns erleichtern, wenn wir uns am Ende des Projekts mit der sich schon androhenden Matheaufgabe herumschlagen müssen. Aber nun erst einmal ein Blick darauf, wie es zwischenzeitlich ausschaut (siehe 4.12.)

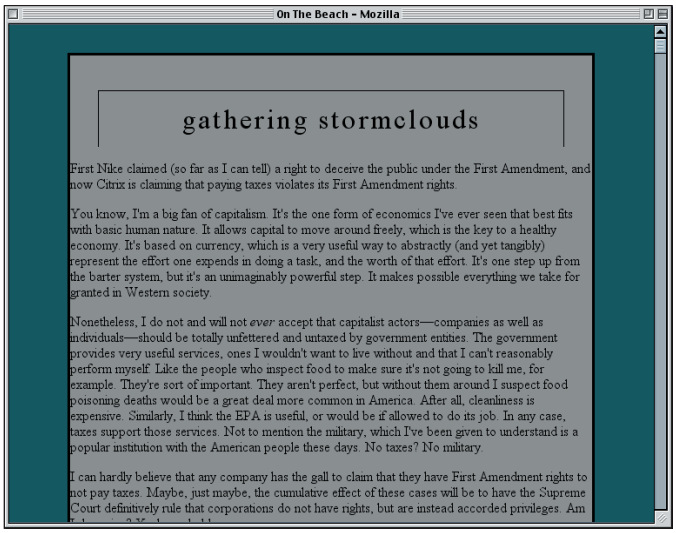


ABBILDUNG 4.12

Durch die zusätzlichen Ränder, Padding und Rahmen wird der Titel hervorgehoben.

4.4.3 Styles für den Inhalt

Es sollte nun nicht zu schwer sein, den Text des Hauptteils in eine Linie mit dem Titel zu bringen. Der erste Schritt ist ganz einfach: Wir werden dem Inhalts-`div`, das den gesamten Text umschließt, einen Rahmen geben.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;
  margin: 1.25em 1em 0; padding: 0.5em 0.25em;
  border: 1px solid black; border-bottom: none;}
#content {border: 1px solid black;}
</style>
```

Das reicht natürlich noch nicht aus. Wenn wir alles nun so ließen, wäre das Inhalts-`div` so breit wie das Haupt-`div`, und die Kanten wären nicht an den Kanten des Titels ausgerichtet. Darum kriegt der Inhalts-`div` noch ein paar Ränder. Aber wie groß sollen die sein?

Tja, der `h1` hat einen oberen Rand mit der Höhe 1,25 em und an der Seite 1 em breit bekommen. Denken Sie jedoch daran, dass der `font-size`-Wert für `h1` `2em` beträgt. Weil seine em-basierten Ränder unter Berücksichtigung des ermittelten `font-size` berechnet sind, bedeutet das, dass wir die Werte für Ränder in unserem Inhalts-`div` verdoppeln müssen. Das führt also zu 2 em Rand links und rechts und einem unteren Rand mit 2,5 em. Weil wir wollen, dass der Rahmen des Inhalts genau oben mit dem Titel abschließt, heißt das: kein oberer Rand.

```
#content {margin: 0 2em 2.5em;
  border: 1px solid black;}
```

Und bei dem Titel wollen wir nicht, dass der inhaltliche Text zu dicht an den Rahmen drum herum kommt. Wir könnten das Padding vom Titel verdoppeln, aber stattdessen wollen wir lieber dem Inhalts-`div` ein großzügiges Padding von anderthalb em spendieren.

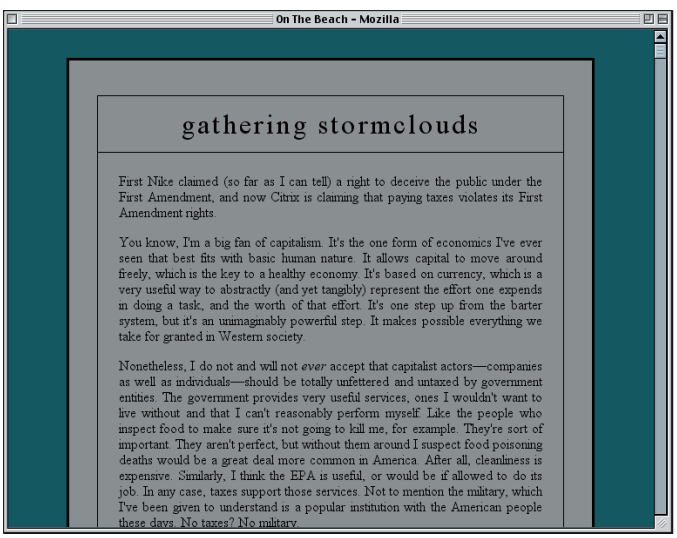
```
#content {margin: 0 2em 3em; padding: 1.5em;
  border: 1px solid black;}
```

Zum Schluss wollen wir dem Text des Inhalts auch noch ein paar Styles angedeihen lassen. Als Erstes werden wir die oberen Ränder aller Absätze hinauswerfen; damit wird der erste Absatz so dicht oben am Inhalts-`div` sein, wie das Padding des `div` es erlaubt, und trotzdem werden alle Absätze einen Abstand von 1 em halten. Und der Text bekommt auch insgesamt einen Blocksatz – siehe Abbildung 4.13.

```
#content {margin: 0 2em 3em; padding: 1.5em;
  border: 1px solid black;}
#content p {margin: 0 0 1em; text-align: justify;}
</style>
```

ABBILDUNG 4.13

Haupttext und Titel integriert



Nun genug der Vorarbeit – wir können uns den Hintergrundbildern zuwenden. Machen wir uns an die Arbeit!

4.4.4 Die Hintergründe werden eingefügt

Nach einem Blick auf das Design in Abbildung 4.13 und den uns zur Verfügung stehenden Bildern werden wir wie folgt vorgehen: Das Haupt-`div` bekommt das Ausgangsbild (`storm-base.jpg`), der Titel kriegt das verblasste und das Inhalts-`div` das abgedunkelte Bild. Dadurch wird auch die Änderung der Textfarbe auf etwas heller nötig sein, aber dem wenden wir uns gleich zu. Hier ist zuerst einmal der Hintergrund des Haupt-`div`s.

```
#main {width: 600px; margin: 2em auto;
  border: 3px solid black;
  background: rgb(7%,13%,15%) url(storm-base.jpg) 0 0 no-repeat;}
```

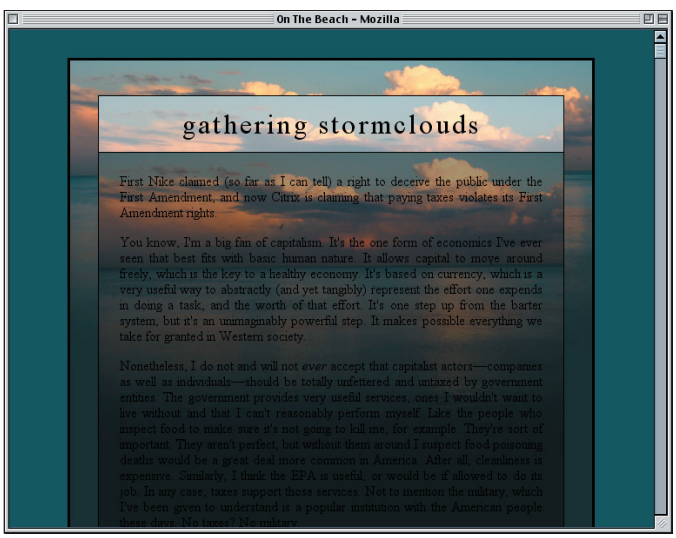
Als Nächstes bekommt der Titel einen Hintergrund.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;
  margin: 1.25em 1em 0; padding: 0.5em 0.25em;
  border: 1px solid black; border-bottom: none;
  background: url(storm-wash.jpg) 0 0 no-repeat;}
```

Schließlich soll auch der Inhalts-`div` nicht ohne Hintergrund bleiben.

```
#content {margin: 0 2em 2.5em; padding: 1.5em;
  border: 1px solid black;
  background: rgb(4%,8%,9%) url(storm-fade.jpg) 0 0 no-repeat;}
```

Den Aufmerksamen unter Ihnen wird das Problem nicht entgangen sein: Alle diese Elemente setzen den Hintergrund so, dass er in der oberen linken Ecke startet, aber die Elemente sind nicht aneinander ausgerichtet. Das bedeutet, die Hintergründe machen das auch nicht – wie (un)schön in Abbildung 4.14 zu sehen ist.



Also müssen wir nun die Offsets herausfinden, mit denen wir die Hintergrundbilder auf Linie bekommen. Das Haupt-`div` können wir natürlich in Ruhe lassen, also machen wir uns zuerst an den Titel. Das ist eigentlich ganz simpel. Wir wissen, dass er einen oberen Rand von 1,25 em und einen linken Rand von 1 em hat, also werden wir uns nach diesen Offsets richten, weil die em für die Ränder die gleichen em sind, die für die Offsets des Hintergrundes verwendet werden.

Doppelnull oben links

Die Hintergrundposition `0 0` ist gleichbedeutend mit `top left`. Wir verwenden hier Zahlen, weil wir später einige von ihnen in negative Werte umzuwandeln haben und Schlüsselwörter und Zahlen nicht kombinierbar sind. Also ist etwas wie `left 50px` in CSS verboten, aber `0 50px` nicht. (In CSS2.1 wurde diese Einschränkung entfernt, aber einige Browser richten sich immer noch danach.)

ABBILDUNG 4.14

Die Hintergründe treten in Erscheinung, aber stehen noch nicht passend.

```
#main h1 {font: 2em/1em "Times New Roman", serif; letter-spacing: 0.1em;
  text-transform: lowercase; text-align: center;
  margin: 1.25em 1em 0; padding: 0.5em 0.25em;
  border: 1px solid black; border-bottom: none;
  background: url(storm-wash.jpg) -1em -1.25em no-repeat;}
```

Beim Inhalts-`div` ist die Situation schon ein wenig komplizierter. Das horizontale Offset ist leicht: Es beträgt `2em`, genauso viel wie der linke Rand. Für das vertikale müssen wir allerdings herauskriegen, wie weit es von der oberen Kante des Inhalts-`div`s bis zur oberen Kante vom Haupt-`div`s ist.

Der Großteil dieser Entfernung wird durch das `h1`-Element mit Beschlag belegt, also sollten wir erst einmal das herausfinden. Innerhalb seines eigenen Referenzrahmens ist der Inhalt von `h1` effektiv 1 em groß und hat oben und unten ein Padding von 0,5 em. Es hat auch einen oberen Rand von 1,25 em (und keinen unteren Rand), und alles zusammen ergibt das 3,25 em Höhe. Sie erinnern sich – das müssen wir wegen des verdoppelten `font-size` des `h1` ebenfalls mal zwei nehmen. Darum ist bezogen auf die ems des Inhalts-`div`s das `h1` 6,5 em hoch. Mit dieser Information und passender hellroter Farbe können wir die Regel entsprechend ändern.

```
#content {margin: 0 2em 2.5em; padding: 1.5em;
  border: 1px solid black;
  color: rgb(210,185,150);
  background: rgb(4%,8%,9%) url(storm-fade.jpg) -2em -6.5em no-repeat;}
```

Position und Ordnung

Uns ist die Reihenfolge nicht durcheinander gekommen: Die Längen- und Prozentwerte für `background-position` werden immer vor dem horizontalen Offset eingestellt, danach folgt dann der vertikale Offset. Darum wird mit `-1em -1.25em` das Bild 1 em nach links und 1,25 em nach oben ausgerichtet. Bei Schlüsselwörtern ist die Reihenfolge nicht so wichtig; `top left` und `left top` sind gleichwertig, obwohl Letzteres in den Netscape 6.x-Versionen Bugs auslösen kann.

Das wär's dann wohl gewesen – oder etwa nicht? Eine Sache haben wir noch vergessen: die oberen Ränder des `h1`-Elements und des Inhalts-`div`s, die dem Hintergrundbereich des Inhalts-`div`s einen Schubs von 2 Pixeln nach unten geben. Tatsächlich hat der Hintergrund von `h1` auch einen Offset von einem Pixel, aber das lassen wir durchgehen, weil es kaum merklich ist.

Wie sich herausstellt, hat der Explorer diese beiden Pixel auch vergessen. Dank der Art, wie er die Hintergrundbilder platziert, sind die Hintergründe bereits ausgerichtet. Wenn wir also nun den Offset ändern, um die Ausrichtung in CSS-konformen Browsern zu reparieren, wäre die Ausrichtung im Explorer sowieso hinausgeworfen worden. Wir machen also Folgendes: Wir lassen die Regel so, wie sie ist, und machen eine neue, die vom Explorer nicht verstanden wird, wohl aber von den aktuelleren Browsern.

```
#content {margin: 0 2em 2.5em; padding: 1.5em;
  border: 1px solid black;
  color: rgb(210,185,150);
  background: rgb(4%,8%,9%) url(storm-fade.jpg) -2em -6.5em no-repeat;}
html>body #content {margin-top: -2px;}
#content p {margin: 0 0 1em; text-align: justify;}
```

Wie in der ersten Hälfte des Projekts wird der Explorer einfach die neu hinzugefügte Regel übergehen. Andere Browser wie Safari, Opera und Gecko-basierte Browser wie Mozilla werden diese Regel verstehen und anwenden. Also haben wir damit ein gutes browserübergreifendes Layout bekommen – siehe Abbildung 4.15.

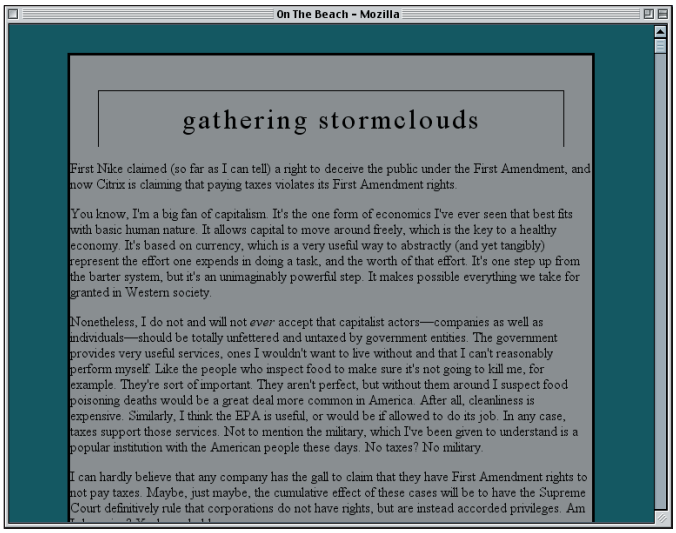


ABBILDUNG 4.15

Die Hintergründe sind nun schön passend ausgerichtet.

4.4.5 Für und Wider

So mächtig wie diese Technik visuell sein kann, sorgt sie doch für einen großen Dateiumfang der Seiten, die sonst andernfalls viel schlanker wären. Designs wie dieses erfordern, dass neben HTML und CSS auch mehrere Hintergründe geladen werden, von denen die meisten nur zum Teil sichtbar sind. Somit könnte man einen Teil der für den Download dieser Seite erforderlichen Bandbreite als verschwendet betrachten, weil darin Bildbereiche enthalten sind, die der User nicht sehen wird.

Das ist zugegebenermaßen ein Nachteil, der bedacht werden sollte. In den für dieses Projekt verwendeten Beispielen waren die Hintergrundbilder nicht sonderlich groß und die abgedunkelten und verblassten Versionen hoch komprimiert, was die Seitengröße niedrig gehalten hat. Ein Design, das auf fünf Variationen eines Hintergrundbildes zurückgreift, von denen alle 800 x 600 oder größer sind, würde zu einer immens hohen Seitengröße führen!

Werden ganze Bilder heruntergeladen und so platziert, wie wir das hier gemacht haben, bringt das einen Vorteil mit sich: Wenn im Dokument die Textgröße angepasst wird, bleibt der Transparenzeffekt immer erhalten (außer wenn Text auf unerwartete Weise umgebrochen wird). Wenn beispielsweise beim »Stormclouds«-Teil der Titeltext über zwei oder mehr Zeilen geht, wird die Ausrichtung des Inhalts-`div`s um einiges aus der Bahn geworfen. Theoretisch könnten wir das Problem über fest fixierte Hintergründe vermeiden, aber der Explorer für Windows unterstützt solche Techniken nicht.

Wir könnten die Ausrichtungsprobleme, die durch Textumbrüche entstehen, mit einer kleinen Änderung in der Struktur umgehen. Statt das Inhalts-`div` hinter das `h1` zu stellen, könnten wir das `h1` in das Inhalts-`div` hineinstellen. Dadurch – und mit einer

Layout mit festem Hintergrund

Mehr Informationen über Techniken mit fixed-attachment-Layouts können Sie im Projekt 12 von *Eric Meyer on CSS* finden.

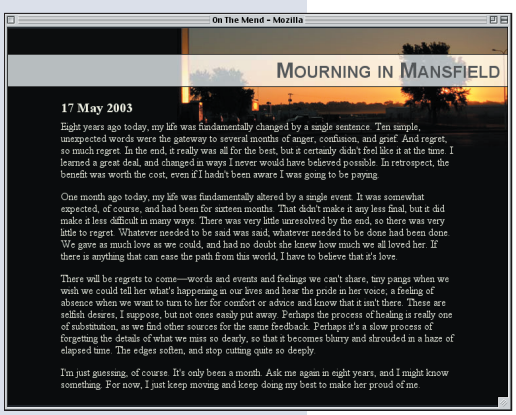
passenden Neuordnung des CSS – kann eine falsche Ausrichtung verhindert werden, wenn der Titeltext am Ende über mehrere Zeilen verteilt wird.

Ein viel einfacherer und schlankerer Ansatz wäre (wenigstens in Fällen, wo Sie einen Hintergrund aufhellen oder abdunkeln wollen), halbtransparente PNG-Grafiken für die Elemente zu nehmen, die verdunkelt oder erhellt werden sollen. Problematisch wird es – wieder – beim Explorer für Windows, der zwar PNG-Bilder unterstützt, aber nicht den Alpha-Kanal, was die Transparenz wieder zunichte macht. Stattdessen zeigt der Explorer nur eine ganz transparente Version des PNG. Es ist auch so, dass Sie Effekte mit Aufhellen oder Abdunkeln wirklich nur bei PNGs erzielen können, was die Auswahlmöglichkeit der visuellen Effekte einschränkt. Mit dem hier vorgestellten Ansatz der Offset-Ausrichtung können Sie ein Ausgangsbild nehmen, eins mit den Bildkonturen und ein weiteres mit einem Riffelglas-Effekt.

Auf jeden Fall ist es wichtig, die in diesem Projekt ausprobierten Techniken mit Umsicht einzusetzen. Sie sind am ehesten nützlich bei Bereichen wie Mastheads, Abschnittsüberschriften oder anderen Stellen, wo der Inhalt knapp ist und der zu dekorierende Bereich sowohl breit als auch kurz ist.

4.5 Spielwiese

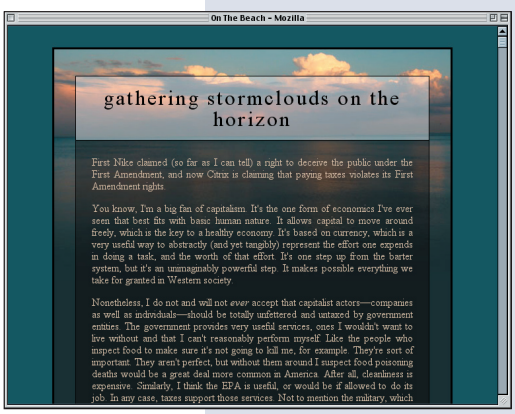
Versuchen Sie, durch das Erstellen von Styles jede der hier beschriebenen Variationen zu erreichen. Wenn das Markup geändert werden muss, damit die Variation erreicht oder überhaupt erst möglich wird, ist das im Text angegeben.



1. Versuchen Sie, bei dem »Mansfield«-Design die Hintergründe (und Hintergrundfarben) auszutauschen. Sie könnten beispielsweise den verblassten Hintergrund für den Titel nehmen, das dunkle Bild als Masthead und das Ausgangsbild für den Body. Vergessen Sie nicht, auch die Farben zu ändern.

2. Ändern Sie im »Stormclouds«-Design die Platzierung der Hintergrundbilder, so dass alle innerhalb ihrer Elemente horizontal zentriert sind. Achten Sie darauf, dass sie immer noch aneinander ausgerichtet bleiben.

3. Ändern Sie das »Stormclouds«-Design so, wie wir es im Abschnitt »Für und Wider« besprochen haben, modifizieren Sie das Markup leicht und arrangieren Sie die Styles neu, so dass das visuelle Ergebnis gleich bleibt, aber das Layout gegen Textumbrüche im Titel nicht mehr so anfällig ist.



5

LISTENBASIERTE MENÜS

Bei euch in Amerika haben sogar Menüs die Gabe der Sprache ...

Ach, diese Menüs! In Amerika sind sie Poesie.

Laurie Lee

In der Design-Community von CSS hat es eine Tendenz gegeben, unsortierte Listen einzusetzen, in der – na ja, im Prinzip alles enthalten sein konnte. Obwohl dies manchmal ziemlich ausuferte, ist es heute eine weit verbreitete Technik, Linksammlungen (manchmal auch Menüs genannt) innerhalb von Listen zu platzieren, wobei jeweils ein Listenelement einen Link bekommt.

Warum ist das eine so beliebte Herangehensweise? Dafür gibt es verschiedene Gründe. Der wichtigste ist, dass es in vielerlei Hinsicht sehr sinnvoll sein kann, wenn Sie eine Liste von Links haben, diese in eine Liste zu legen. Aus Sicht der Semantik im Markup ist das ziemlich naheliegend.

Aus der Sicht des Stylings gibt es auch eine Reihe Vorteile. Weil jedes Listenelement einen Link enthält, können zwei unterschiedliche Elemente (`li` und `a`) unabhängig voneinander gestylt werden. Weil die Elemente die Basis für das Styling sind, ist es gut, möglichst viele Elemente zu haben, mit denen man arbeiten kann.

submental mutterings

Other Mutters

- 13 September 2002
- 6 September 2002
- 25 October 2002
- 8 November 2002
- 14 November 2002
- 17 November 2002
- 3 December 2002
- 4 December 2002

14 November 2002

Early this afternoon, my paternal grandfather laid down for his habitual afternoon nap, and never woke up. All things considered, it was a gentle passing for a gentle man.

When I was in my early teens, he and my grandmother took me on a trip to Churchill, Manitoba, on the Hudson Bay. It was their tradition to take each grandchild on a trip at that approximate age, and when my turn came up they were headed to the Hudson Bay—so I went along. We drove across the upper Midwest into Canada, and flew up to Churchill. It's pretty cold up there, and we were spending a lot of time outdoors as my grandparents sought out bird species they'd never seen before to add to their life lists. I saw a lot of tundra that summer, which was kind of boring. I also saw the rusted hulk of a beached ship, which interested me a lot more.

One day we took a break from the birding to climb into a small boat and head out into the mouth of the river that feeds into the Hudson. The goal was to see Beluga whales swimming. We motored out into river, and it was terribly cold. The temperature was low enough, but the strong wind knifed through me like I wasn't wearing anything. When the bow started to spray and get us wet, the situation was just that much worse. My hands hurt from the cold, and I huddled in the boat. No doubt I was

submental mutterings

Other Mutters

13 September 2002
6 September 2002
25 October 2002
8 November 2002
14 November 2002
17 November 2002
3 December 2002
4 December 2002

14 November 2002

Early this afternoon, my paternal grandfather laid down for his habitual afternoon nap, and never woke up. All things considered, it was a gentle passing for a gentle man.

When I was in my early teens, he and my grandmother took me on a trip to Churchill, Manitoba, on the Hudson Bay. It was their tradition to take each grandchild on a trip at that approximate age, and when my turn came up they were headed to the Hudson Bay—so I went along. We drove across the upper Midwest into Canada, and flew up to Churchill. It's pretty cold up there, and we were spending a lot of time outdoors as my grandparents sought out bird species they'd never seen before to add to their life lists. I saw a lot of tundra that summer, which was kind of boring. I also saw the rusted hulk of a beached ship, which interested me a lot more.

One day we took a break from the birding to climb into a small boat and head out into the mouth of the river that feeds into the Hudson. The goal was to see Beluga whales swimming. We motored out into river, and it was terribly cold. The temperature was low enough, but the strong wind knifed through me like I wasn't wearing anything. When the bow started to spray and get us wet, the situation was just that much worse. My hands hurt from the cold, and I huddled in the boat. No doubt I was

submental mutterings

Other Mutters

13 September 2002
6 September 2002
25 October 2002
8 November 2002
14 November 2002
17 November 2002
3 December 2002
4 December 2002

14 November 2002

Early this afternoon, my paternal grandfather laid down for his habitual afternoon nap, and never woke up. All things considered, it was a gentle passing for a gentle man.

When I was in my early teens, he and my grandmother took me on a trip to Churchill, Manitoba, on the Hudson Bay. It was their tradition to take each grandchild on a trip at that approximate age, and when my turn came up they were headed to the Hudson Bay—so I went along. We drove across the upper Midwest into Canada, and flew up to Churchill. It's pretty cold up there, and we were spending a lot of time outdoors as my grandparents sought out bird species they'd never seen before to add to their life lists. I saw a lot of tundra that summer, which was kind of boring. I also saw the rusted hulk of a beached ship, which interested me a lot more.

One day we took a break from the birding to climb into a small boat and head out into the mouth of the river that feeds into the Hudson. The goal was to see Beluga whales swimming. We motored out into river, and it was terribly cold. The temperature was low enough, but the strong wind knifed through me like I wasn't wearing anything. When the bow started to spray and get us wet, the situation was just that much worse. My hands hurt from the cold, and I huddled in the boat. No doubt I was

5.1 Projektziele

Nachdem wir uns mit verschiedenen Kundenprojekten beschäftigt haben, scheint es nun an der Zeit zu sein, dass wir uns eine Atempause gönnen und uns ein persönliches Projekt vornehmen. Dafür werden wir daran arbeiten, die Präsentation der Links in der Seitenleiste eines Tagebuchs aufzubessern. Wir wollen ein paar grundlegende Designschwerpunkte definieren und schauen, wohin wir damit kommen.

- Die Seite, mit der wir beginnen, hat schon ein paar Styles, also müssen die Styles des Menüs zu der schon vorhandenen Präsentation passen.
- Die Links im Menü sollen optisch voneinander getrennt sein; das heißt, unsere Linkliste soll nicht ohne Trennlinien oder andere visuelle Effekte bleiben.
- Am Ende soll ein Design herauskommen, bei dem das Menü offen und luftig erscheint, so dass die Links im Design als Teil des anderen Inhalts der Seite wirken.
- Wir werden uns ein weiteres Design ausdenken, bei dem die Links in einer Box liegen oder einer anderen visuellen Einheit liegen, damit sie eindeutig vom Hauptinhalt getrennt wirken.

Das sind nun unsere Ziele, und wir beginnen mit dem Stylen!

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

5.2 Vorbereitungen

Laden Sie die Dateien für Projekt 5 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch05proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und erneut laden.

Der zweite Auftritt

Wenn Sie *Eric Meyer on CSS* kennen, werden Sie das Design dieses Projekts wiedererkennen, weil es das Projekt 9 über »Mehrspaltiges Layout« aus jenem Buch ist. Der Unterschied hier besteht darin, dass die Links nun in einer unsortierten Liste enthalten sind und dass es ein Tagebucheintrag mit einem anderen Datum ist.

5.3 Die Grundlagen

Weil wir uns darauf konzentrieren wollen, wie man eine unsortierte Linkliste innerhalb eines existierenden Designs stylen kann, gehen wir davon aus, dass schon etwas CSS vorhanden ist. Das Ausgangs-Stylesheet sehen Sie in Listing 5.1 und der Abbildung 5.1.

Listing 5.1 Die Ausgangs-Styles

```
<style type="text/css">
html {margin: 0; padding: 0;}
body {font: 80% Verdana, Arial, Helvetica, sans-serif;
   margin: 0; padding: 0;
   background: rgb(95%,95%,80%); color: black;}
```

```
h1 {font-size: 200%; text-transform: lowercase; letter-spacing: 3px;
   margin: 0; padding: 0.66em 0 0.33em 29%;
   background: rgb(85%,85%,70%);}
h3 {font-size: 1.33em; margin: 0; padding: 0;
   border-bottom: 1px solid black;}
h4 {font-size: 1em; margin: 0; padding: 0.33em 0 0;
   border-bottom: 1px solid rgb(50%,50%,35%);}
h1, h3, h4 {line-height: 1em;}
p {line-height: 1.5; margin: 0.5em 0 1em;}
div#entry {margin: 2em 10% 1em 30%; padding: 0;}
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;}
</style>
```

ABBILDUNG 5.1
Das vorhandene Design

Im Wesentlichen müssen wir die Liste mit den Links neu stylen, damit sie besser zum Rest des Designs passt. Im Verlauf dieser Arbeit werden wir uns für die gleiche Liste ein paar andere Möglichkeiten der Präsentation anschauen.

5.3.1 Prüfung des Markups

Um unsere Styles gut planen zu können, sollten wir uns das Markup anschauen, das die Links enthält. Sie finden es in Listing 5.2.

Listing 5.2 Das Markup für die Links

```
<div id="sidebar">
<h4>Other Mutters</h4>
<ul>
<li><a href="mutter01.html">13 September 2002</a></li>
<li><a href="mutter02.html">6 September 2002</a></li>
<li><a href="mutter03.html">25 October 2002</a></li>
```

Explorer springt

Wenn Sie mit dem Internet Explorer für Windows an diesem Projekt arbeiten, fällt Ihnen möglicherweise ein kleiner »Versatz« in dem Text im Float der Seitenleiste auf. Das liegt an einem Bug im Internet Explorer, aber den können wir glücklicherweise umgehen. Wir haben hier nicht genug Platz, das näher auszuführen, aber Sie erfahren alle Details bei `http://www.positioniseverything.net/explorer/threepxtest.html`.

```
<li><a href="mutter04.html">8 November 2002</a></li>
<li><a href="mutter05.html">14 November 2002</a></li>
<li><a href="mutter06.html">17 November 2002</a></li>
<li><a href="mutter07.html">3 December 2002</a></li>
<li><a href="mutter08.html">4 December 2002</a></li>
</ul>
</div>
```

Das ist alles ganz simpel gestrickt: eine einfache, unsortierte Liste mit einem Link in jedem Listenelement. Hier liegt also keine große Überraschung begraben, da wir ja die Abbildung 5.1 kennen, aber es war es wert zu schauen, was wir hier überhaupt vorfinden.

Als Erstes wollen wir die Gliederungspunkte (*Bullets*) und die Einrückung der Liste selbst loswerden. Weil das über alle Designideen, die wir ausprobieren wollen, gleich bleiben wird, erledigen wir es gleich jetzt, um es vom Hals zu haben.

```
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;}
#sidebar ul {list-style: none; margin: 0; padding: 0;}
</style>
```

Indem wir sowohl Ränder als auch Padding komplett entfernen, sind wir den Einrückungseffekt los, den Listen so mit sich bringen. Weil einige Browser diese Einrückung über Ränder vornehmen und andere das mit Padding erledigen, sind wir auf sicherem Boden, wenn wir beides auf Null setzen (siehe Abbildung 5.2).

ABBILDUNG 5.2

Die Bullets der Liste und die Einrückung sind verschwunden.

Jetzt können wir verschiedene Designideen für unsere Linkliste ausprobieren.

5.4 Offen und luftig

Für die erste Version werden wir Styles erstellen, durch die die lockerer angeordneten Links sichtbar getrennt sind. Dafür werden wir die Links vertikal auseinander schieben und dann Trennungen zwischen die Links setzen.

5.4.1 Trennungen

Zuerst werden wir die Links auseinander ziehen, indem wir den Listenelementen, in denen sie enthalten sind, Padding zugeben. Oben und unten – das sollte dann schon reichen.

```
#sidebar ul {list-style: none; margin: 0; padding: 0;}
#sidebar li {padding: 0.5em 0 0.25em;}
</style>
```

Mehr brauchen wir nicht, um die Links auseinander zu ziehen. Das Schöne an diesem Vorgehen ist, dass falls wir jemals beschließen sollten, die Links müssten doch weiter auseinander sein (oder dichter zusammen), dann brauchten wir nur noch den Wert für `padding` zu ändern.

Und jetzt zu den Trennlinien. Dafür brauchen wir einfach nur jedem Listenelement einen unteren Rahmen zu geben.

```
#sidebar li {padding: 0.5em 0 0.25em;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

Indem wir einen soliden Rahmen mit einer Mischfarbe aus den Schattierungen des Hintergrunds und des unteren Rahmens der Überschrift »Other Mutters« nehmen, können wir einen schönen subtilen Effekt schaffen (siehe Abbildung 5.3).

Abbildung 5.3

Durch die Rahmen an den Listenelementen werden die Links optisch getrennt.

Ungleiches Padding?

Wir haben das Padding absichtlich ungleich gemacht. Wegen der Art und Weise, wie englischer Text geformt ist, ist im Allgemeinen unter einer Textzeile mehr sichtbarer Raum als darüber. Darum haben wir oben mehr Padding angegeben als unten. Das ist eine Entscheidung des Designs. Andere sind auch möglich, und in Situationen, bei denen der Text in Großbuchstaben erscheint, sollte das Padding oben und unten wahrscheinlich eher symmetrisch sein.

Farbmischung

Die Farbschattierungen in diesem Design lassen sich mit dem *Color Blender* herausfinden (`http://www.meyerweb.com/eric/tools/color-blend/`).

Die Links haben nun einen passenden Abstand und eine Trennung, aber um die Farben sollten wir uns noch etwas kümmern; dieses Blau passt einfach nicht gut zu den schönen Erdtönen, die wir beim restlichen Design verwenden. Und Tschüss auch für die Unterstreichungen! Das machen wir gleich mal als Erstes.

```
#sidebar li {padding: 0.5em 0 0.25em;
  border-bottom: 1px solid rgb(84%,84%,69%);}
#sidebar a {text-decoration: none;}
</style>
```

Jetzt brauchen wir noch ein paar Farben, die gut den Rest des Designs ergänzen. Grün und Beige passen oft gut zusammen, also probieren wir es mal mit dieser Kombination.

```
#sidebar a {text-decoration: none;}
#sidebar a:link {color: rgb(20%,40%,0%);}
</style>
```

ABBILDUNG 5.4

Besuchte und nicht besuchte Links bekommen passendere Farben.

Das ist eine ziemlich gute Entscheidung, also nehmen wir eine Variante der Farbe für die besuchten Links. Diese Links sollen eine Farbe zwischen `rgb(20%,40%,0%)` und der Body-Hintergrundfarbe `rgb(95%,95%,80%)`bekommen. Das können Sie gut in Abbildung 5.4 sehen, wo der Link für den aktuellen Eintrag die Farbe für den besuchten Link hat. (Na, es ist doch auch die aktuelle Seite – also ist sie ja besucht worden!)

```
#sidebar a:link {color: rgb(20%,40%,0%);}
#sidebar a:visited {color: rgb(58%,68%,40%);}
</style>
```

Mehr brauchen wir im Grunde nicht machen, damit diese Links zum Design passen. Natürlich ist das nur ein Anfang; es gibt beinahe unzählige Arten, wie wir das Markup ändern können, um dem Design zu genügen.

5.4.2 Pfeil-Styles

Um die Seitenleiste ein wenig aufzupeppen, wollen wir eine Pfeil-Grafik einfügen. Dafür müssen wir die Styles der Seitenleiste anpassen, aber wie Sie sehen werden, können wir das leicht bewerkstelligen. Wir werden nun der unteren rechten Ecke des Seitenleisten-Hintergrunds ein Pfeil-Bild zufügen.

```
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;
  background: url(arrow.gif) 100% 100% no-repeat;}
```

Nun müssen wir uns darum kümmern, dass der Pfeil auch genug Platz hat, um über den unteren Bereich des letzten Listenelements hinauszuragen. Dafür sollte etwas Padding unten reichen, das genau der Höhe des Bildes entspricht (siehe Abbildung 5.5).

```
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;
padding: 0 0 15px;
  background: url(arrow.gif) 100% 100% no-repeat;}
```

ABBILDUNG 5.5

Die untere rechte Ecke der Seitenleiste bekommt einen Pfeil.

Uns springt gleich das nächste Problem ins Auge: Die Trennungslinien der Links ragen über die Mittellinie des Pfeils, das geht also nicht. Wir müssen den rechten Rand der Links ein Stück wegschieben, und wenn wir den Pfeil genauer untersuchen, sehen wir, dass sich 6 Pixel zwischen seiner rechten Kante und der Mittellinie befinden. Also bekommt die Liste einfach einen Rand in genau dieser Länge. Wenn wir das so machen, müssen wir auch den `h4` für »Other Mutters« anpassen, damit sein Rand der gleiche wie bei der Liste ist.

```
#sidebar ul {list-style: none; margin: 0; padding: 0;}
#sidebar h4, #sidebar ul {margin: 0 6px 0 0;}
#sidebar li {padding: 0.5em 0 0.25em;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

Ein Fall fürs Gruppieren

Wir haben eine Regel für einen Gruppen-Selektor erstellt, weil darüber spätere Anpassungen erleichtert werden. Wenn wir die Ränder für `h4` und `ul` einzeln einstellen, müssen wir zum Anpassen ihrer Ränder immer beide Regeln editieren. Aber so kümmert sich eine Regel um beides.

Nun werden wir die Links und den Pfeil verbinden. Gut, dass die Pfeil-Grafik mit einer drei Pixel breiten Zentrallinie konstruiert wurde; so brauchen wir nur einen Rahmen mit der entsprechenden Breite, Farbe und dem passenden Style angeben.

```
#sidebar ul {list-style: none; margin: 0; padding: 0;
  border-right: 3px double rgb(50%,50%,35%);}
```

Wir sehen gleich, dass der Pfeil ein wenig zu dicht an den Links steht – das sieht irgendwie eingezwängt aus. Darum werden wir den Rand der Liste vergrößern und den Pfeil nach unten schieben, indem wir der Liste unten Padding zugeben (siehe Abbildung 5.6).

```
#sidebar ul {list-style: none; margin: 0; padding: 0 0 10px;
  border-right: 3px double rgb(50%,50%,35%);}
```

ABBILDUNG 5.6

Der Rahmen der Liste verschmilzt mit dem Hintergrundbild der Seitenleiste.

Wie funktioniert das? Weil das Hintergrundbild innerhalb des Paddings der Seitenleiste sitzt und die Liste ebenfalls im gleichen Padding ist, wird jedes Aufstocken der Höhe der Liste (z.B. durch zusätzliches Padding) die Höhe der Seitenleiste steigern und darum das Pfeil-Bild nach unten schieben. Das untere Padding erweitert obendrein den rechten Rahmen der Liste, also bleibt es optisch mit dem Hintergrundbild verschmolzen.

5.4.3 Die Links werden gestylt

Die Styles für den Pfeil sind schon ganz schmuck, aber fertig sind wir noch lange nicht. Wir wollen den Text der Links nach rechts hinüberschieben und darauf basierend einiges abändern. Zuerst richten wir den Text aus.

```
#sidebar ul {list-style: none; margin: 0; padding: 0 0 10px;
  border-right: 3px double rgb(50%,50%,35%);
  text-align: right;}
```

Damit wird der Text komplett gegen den rechten Rahmen gepresst – das wollen wir nicht wirklich. Wir könnten einen Rand bei den Listenelementen einstellen, aber das würde den unteren Rahmen vom doppelten Rahmen rechts trennen, ist also passé. Wir könnten den Listenelementen auch rechts Padding zugeben, aber stattdessen wollen wir lieber bei den Links selbst rechtes Padding einfügen.

```
#sidebar a {text-decoration: none; padding: 0 0.5em 0 0;}
```

Wozu das Padding? Weil unser nächster Schritt darin bestehen wird, die Rahmen der Listenelemente zu entfernen. Für den Moment werden wir das einfach durch ein Auskommentieren der Deklaration `border-bottom` vornehmen.

```
#sidebar li {padding: 0.5em 0 0.25em;
  /* border-bottom: 1px solid rgb(84%,84%,69%); */}
```

Sie fragen sich wohl schon, warum wir die Trennlinien wegnehmen, die wir gerade erst erstellt haben? Weil wir dem Problem auf eine andere Weise beikommen werden. Statt die Listenelemente einzurahmen, werden wir die Links selbst mit Rahmen versehen. Das Ergebnis können Sie in Abbildung 5.7 sehen.

```
#sidebar a {text-decoration: none; padding: 0 0.5em 0 0;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

ABBILDUNG 5.7

Eine interessante Variation: Die Rahmen werden zu den Links verlagert.

Kommt Ihnen die letzte Änderung bekannt vor? Das sollte sie: Der Rahmen-Style, den wir bei den Links angewendet haben, ist der gleiche, den wir gerade bei der Regel für die Listenelemente auskommentiert haben.

Es gibt offensichtliche Unterschiede zwischen den beiden Vorgehensweisen. Als die Rahmen noch bei den Listenelemente waren, verliefen sie von einer Seite der Seitenleiste bis zur anderen. Das kommt daher, weil Listenelemente Blocklevel-Boxen generieren, ähnlich wie Absätze und `div`s. Die Links andererseits generieren Inline-Boxen,

was bedeutet, dass die Rahmen nur unter dem Text selbst verlaufen werden. Das gilt sogar dann, wenn der Link über mehrere Zeilen umgebrochen wird.

Der Effekt ist immer noch potenziell sehr hilfreich, weil wir über dieses neue Vorgehen die Rahmen anders gestalten können, wenn die Maus darüber fährt. Das ist etwas, das wir im Explorer nicht leisten konnten, als die Listenelemente einen Rahmen hatten. Also werden wir eine Regel einfügen, die sowohl Text als auch Rahmenfarbe ändert, wenn sich ein Link im Rollover-Status befindet.

```
#sidebar a:visited {color: rgb(58%,68%,40%);}
#sidebar a:hover {color: rgb(10%,20%,0%);
  border-color: rgb(98%,48%,40%);}
</style>
```

Dies allein sollte schon reichen, weil der Text dunkler und der Rahmen rötlich wird, wenn der Mauszeiger über einen Link gehalten wird. Das können wir noch weiter ausbauen, indem wir dem Link im Rollover-Status ein Hintergrundbild geben. Wir können das Bild nur in den Link-Hintergrund legen, also werden wir eine Grafik mit einem Halbpfeil nehmen: `arrow2.gif`. Diese wird in der unteren linken Ecke des Links platziert.

```
#sidebar a:hover {color: rgb(10%,20%,0%);
  border-color: rgb(98%,48%,40%);
  background: url(arrow2.gif) 0 100% no-repeat;}
```

So wie bei dem Pfeil der Seitenleiste müssen wir dem Hintergrund genug Padding geben, damit er sichtbar ist und nicht durch Text überlagert wird. Weil wir dieses Padding nur dann brauchen, wenn das Hintergrundbild erscheint, werden wir das Padding des Links nur während des Rollovers aufstocken. Das hat den in Abbildung 5.8 gezeigten Effekt.

ABBILDUNG 5.8

Die Links werden interaktiver, wenn sie einen Rollover-Effekt bekommen.

Beliebiges Rollover

Die CSS-Spezifikation erlaubt es in der Tat, dass Rollover-Styles auf jedes beliebige Element – nicht nur Links – angewandt werden, und die meisten modernen Browser ermöglichen das auch. Zu dieser Gruppe gehört – leider – der Explorer nicht, hier sind die Rollover-Effekte nur auf Links beschränkt. Oder jedenfalls war das noch bis vor kurzem so, da Peter Nederlof nun einen Weg gefunden hat, die Fähigkeiten des Internet Explorer für Windows auszubauen, damit beliebige Elemente für Rollover-Effekte gestylt werden können. Sie können das unter `http://www.xs4all.nl/~peterned/csshover.html` nachlesen.

```
#sidebar a:hover {color: rgb(10%,20%,0%);
  border-color: rgb(98%,48%,40%);
  background: url(arrow2.gif) 0 100% no-repeat;
  padding-left: 15px;}
```

Denken Sie daran, dass diese Effekte nur deshalb gut funktionieren, weil die Links nicht auf mehrere Zeilen umgebrochen werden. In Situationen, bei denen ein Zeilenumbruch wahrscheinlich ist, ist möglicherweise eine andere Vorgehensweise beim Design besser.

Dennoch sollten wir unser Möglichstes tun, um uns vor potenziellen Problemen durch Zeilenumbruch zu schützen. Nehmen wir an, dass der Inhalt eines Links zwischen 0 und 15 Pixel schmaler ist als der Link selbst. In diesem Fall würde das Padding, das beim Rollover angewendet wird, einen Zeilenumbruch verursachen, was bedeutet, dass das Aussehen des Links sich zwischen Rollover- und Nicht-Rollover radikal unterscheiden kann. Wir könnten jeglichen Zeilenumbruch mit `white-space:nowrap` verhindern, aber das könnte dazu führen, dass Links komplett aus der Seitenleiste herausragen, und das ist nicht akzeptabel.

Darum werden wir das Padding mit 15 Pixeln durch ein unsichtbares Äquivalent ersetzen: einen 15-Pixel-Rahmen auf den Links der Seitenleiste.

```
#sidebar a {text-decoration: none;
  padding: 0 0.5em 0 0; margin-left: 15px;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

Wir wollen, dass der linke Rand auf jeden Fall durch das Padding ersetzt wird, wenn bei einem Link in der Seitenleiste ein Rollover geschieht, also müssen wir den Rand im Rollover-Status entfernen.

```
#sidebar a:hover {color: rgb(10%,20%,0%);
  border-color: rgb(98%,48%,40%);
  background: url(arrow2.gif) 0 100% no-repeat;
  padding-left: 15px; margin-left: 0;}
```

Das wird nicht verhindern, dass die Links umgebrochen werden, aber es verhindert den Reflow. Wenn der Link und sein Rand zusammenpassen, gilt das auch für den besuchten Link und sein Padding, wie in gut in Abbildung 5.9 – in der das Browserfenster verkleinert wurde – zu sehen ist.

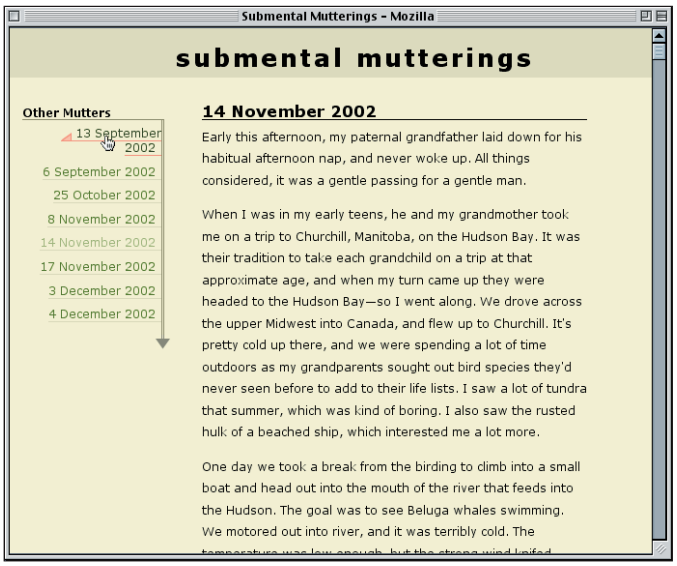


Abbildung 5.9

Wird der Rand gegen Padding eingetauscht, verhindert das, dass die Links unerwartet anders dargestellt werden (Reflow).

Mit dieser Änderung haben wir das Stylesheet aus Listing 5.3.

Listing 5.3 Das Stylesheet der »luftigen Links«

```
html {margin: 0; padding: 0;}
body {font: 80% Verdana, Arial, Helvetica, sans-serif;
   margin: 0; padding: 0;
   background: rgb(95%,95%,80%); color: black;}
h1 {font-size: 200%; text-transform: lowercase; letter-spacing: 3px;
   margin: 0; padding: 0.66em 0 0.33em 29%;
   background: rgb(85%,85%,70%);}
h3 {font-size: 1.33em; margin: 0; padding: 0;
   border-bottom: 1px solid black;}
h4 {font-size: 1em; margin: 0; padding: 0.33em 0 0;
   border-bottom: 1px solid rgb(50%,50%,35%);}
h1, h3, h4 {line-height: 1em;}
p {line-height: 1.5; margin: 0.5em 0 1em;}
div#entry {margin: 2em 10% 1em 30%; padding: 0;}
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%; padding: 0 0 15px;
  background: url(arrow.gif) 100% 100% no-repeat;}
#sidebar ul {list-style: none; margin: 0; padding: 0 0 10px;
  border-right: 3px double rgb(50%,50%,35%);
  text-align: right;}
#sidebar h4, #sidebar ul {margin: 0 6px 0 0;}
#sidebar li {padding: 0.5em 0 0.25em;
  /* border-bottom: 1px solid rgb(84%,84%,69%); */}
#sidebar a {text-decoration: none;
  padding: 0 0.5em 0 0; margin-left: 15px;
  border-bottom: 1px solid rgb(84%,84%,69%);}
#sidebar a:link {color: rgb(20%,40%,0%);}
```

```
#sidebar a:visited {color: rgb(58%,68%,40%);}
#sidebar a:hover {color: rgb(10%,20%,0%);
  border-color: rgb(98%,48%,40%);
  background: url(arrow2.gif) 0 100% no-repeat;
  padding-left: 15px; margin-left: 0;}
```

Wir sind aber noch nicht fertig – dies ist erst die erste Hälfte des Projekts.

5.5 Eingerahmte Links

Nachdem wir mit Styles experimentiert haben, die den Links der Seitenleiste eine offene, luftige Erscheinung geben, wollen wir eine andere Herangehensweise probieren: Die Links sollen in einer Box verpackt und zu schaltflächenähnlichen Objekten gemacht werden. Dafür werden wir zu einer früheren Version des Projekts zurückkehren und von dort aus weitermachen.

5.5.1 Änderungen

Wir werden die Datei replizieren, die wir zur Zeit von Abbildung 5.3 hatten, und fünf Änderungen vornehmen. Zuerst werden wir dem `h4` den Rahmen wegnehmen und seine Textfarbe ändern; die Regel sieht also so aus:

```
h4 {font-size: 1em; margin: 0; padding: 0.33em 0 0;
   color: rgb(50%,50%,35%);}
```

Zweitens werden wir die Seitenleiste vereinfachen, indem wir ihr Hintergrundbild und das Padding entfernen.

```
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;}
```

Als Drittes verändern wir den doppelten Rahmen auf der rechten Seite der Liste auf einen soliden kastenförmigen Rahmen mit einem Pixel Stärke.

```
#sidebar ul {list-style: none; margin: 0; padding: 0 0 10px;
  border: 1px solid rgb(73%,73%,58%);
  text-align: right;}
```

Viertens entfernen wir den linken Rand der Links.

```
#sidebar a {text-decoration: none;
  padding: 0 0.5em 0 0;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

Schließlich reduzieren wir die Rollover-Styles der Links auf einen schlichten Wechsel der Text- und Hintergrundfarben; die Deklarationen, über die die Rahmenfarbe geändert, Padding und ein Hintergrundbild eingefügt wurden, kommen alle weg.

```
#sidebar a:hover {color: rgb(10%,20%,0%);
  background: #FFF;}
```

Das Ergebnis all dieser Änderungen zusammengenommen sehen Sie in Abbildung 5.10.

ABBILDUNG 5.10

Nach einer Reihe Änderungen sieht die Seitenleiste ganz schön anders aus.

5.5.2 Links im Kasten

Wir haben einen Kasten erstellt, der die Links vollkommen umgibt, aber den »kastigen« Look wollen wir noch etwas weiter treiben und alle Links in einen Kasten stellen. Das kriegen wir ganz einfach hin – wir machen den unteren Rahmen zu einem vollständigen Rahmen und passen das Padding an, so dass der Link-Text den Rahmen nicht zu nahe kommt.

```
#sidebar a {text-decoration: none;
  padding: 0 0.25em;
  border: 1px solid rgb(84%,84%,69%);}
```

An diesem Punkt wollen wir eigentlich nicht, dass die Links rechtsbündig sind, weil damit die Rahmen der Links direkt an den für die Liste eingestellten Rahmen stoßen. Stattdessen werden wir sie zentrieren, und damit bekommen wir die Kanten der Links weg vom Rahmen der Liste. Damit der Text von `h4` auch zentriert ausgerichtet wird, werden wir `text-align` auf das Seitenleisten-`div` anwenden und es von der Liste entfernen.

```
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;
  text-align: center;}
#sidebar ul {list-style: none; margin: 0; padding: 0 0 10px;
  border: 1px solid rgb(73%,73%,58%);}
```

Jetzt brauchen wir eine Möglichkeit, wie wir die Links voneinander trennen können. Um das hinzukriegen, wäre es am einfachsten, wenn wir die unteren Rahmen der Listenelemente wieder herstellen und das Padding anpassen, damit es ein wenig ausgewogener ist (siehe Abbildung 5.11).

```
#sidebar li {padding: 0.5em 0;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

Blocks ausrichten

Ein anderer Grund für die Verwendung der Seitenleiste zur Textausrichtung liegt darin, dass die Links selbst Inlinelevel-Elemente sind, und darum kann `text-align` nicht direkt auf sie angewendet werden. `text-align` kann nur auf Blocklevel-Elemente angewendet werden.

ABBILDUNG 5.11
In den Kasten mit den Links

5.5.3 Linien rittlings

Untersucht man die Abbildung 5.11 genauer, kommt einem fast von selbst in den Sinn, dass die Links ja so weit nach unten verschoben werden können, dass sie »auf« den Rahmen der Listenelemente sitzen. Wenn wir das machen, sieht das so aus, als wäre jeder Link wie ein kleines Plakat, das mit einem strammen Draht zwischen den Seiten der Liste aufgehängt ist.

Wir überlegen kurz, ob das auch mit einem negativen Rand hinzukriegen sei, aber leider nicht: Die Listen sind inline, und Ränder oben und unten haben keinen Effekt auf das Layout eines Inline-Elements. Glücklicherweise wirkt sich relative Positionierung auf das Inline-Layout aus, also können wir das als Alternative nehmen.

Wir wissen, dass die Listenelemente nur Links enthalten, und alle haben ein em Padding (ein halbes oben, ein halbes unten). Weil wir alle potenziellen Variablen aus dieser Situation entfernen wollen, werden wir die `line-height` der Listenelemente auf `1em` einstellen, anstatt das der Standardeinstellung des Browsers zu überlassen.

```
#sidebar li {padding: 0.5em 0; line-height: 1em;
  border-bottom: 1px solid rgb(84%,84%,69%);}
```

Jetzt können wir die Links um den entsprechenden Offset verschieben. Dank unserer Werte für `line-height` und `padding` wissen wir, dass jedes Listenelement 2em groß ist. Wir wollen die Links um die Hälfte nach unten schieben, also bedeutet das:

```
#sidebar a {text-decoration: none;
  padding: 0 0.25em;
  border: 1px solid rgb(84%,84%,69%);
  position: relative; top: 1em;}
```

Oben oder unten?

Wir hätten auch `bottom: -1em` nehmen können, um den gleichen Effekt zu erreichen, weil das genau äquivalent zu `top: 1em` ist, wenn ein Element relativ positioniert wird. In solchen Fällen können Sie sich ganz nach Belieben entweder für `top` oder `bottom` entscheiden.

Jetzt ist jeder Link um ein em nach unten verschoben. Das bedeutet, dass die Rahmen der Listenelemente sie genau durchschneiden werden, weil die Links (wie alle Elemente) standardmäßig einen transparenten Hintergrund haben. Wir werden die gleichen Hintergrundfarben nehmen wie für `body` selbst. Das führt uns zu dem Ergebnis aus Abbildung 5.12.

```
#sidebar a {text-decoration: none;
  padding: 0 0.25em;
  border: 1px solid rgb(84%,84%,69%);
  background: rgb(95%,95%,80%);
  position: relative; top: 1em;}
```

Abbildung 5.12

Wir wenden auf die Links relative Positionierung und Hintergründe an.

Das ist schon recht nett, aber es bleibt noch ein kleines Problem: Der letzte Link in der Liste sitzt zu dicht auf dem unteren Rahmen des Kastens. Der einzige Grund, warum er nicht aus der Liste herausragt, ist, dass die Liste unten 10 Pixel Padding hat. Wenn wir das so ändern, dass es exakt der Höhe der Listenelemente entspricht, haben wir genau den Platz, den wir brauchen.

```
#sidebar ul {list-style: none; margin: 0; padding: 0 0 2em;
  border: 1px solid rgb(73%,73%,58%);}
```

An diesem Punkt brauchen wir eigentlich nur noch den Titel der Seitenleiste »Other Mutters« etwas aufzupolieren. Der könnte im Prinzip auch so stehen bleiben, aber er sieht schon merkwürdig aus, wie er da so auf dem Kasten hockt. Die Lösung: Er kriegt seinen eigenen Kasten – einen, der zu dem Rahmen um die Liste herum passt.

```
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;
  text-align: center;}
#sidebar h4 {border: 1px solid rgb(73%,73%,58%);}
#sidebar ul {list-style: none; margin: 0; padding: 0 0 2em;
  border: 1px solid rgb(73%,73%,58%);}
```

Weil nun `h4` einen unteren Rahmen (so wie alle anderen Seiten) hat und die Liste einen oberen Rahmen – und die beiden grenzen genau aneinander –, wird die Illusion entstehen, dass es einen Rahmen aus zwei Pixeln zwischen dem Titel und den Links gibt. Um das loszuwerden, können wir entweder den oberen Rahmen von der Liste entfernen oder den unteren aus der Überschrift. Weil wir sowieso gerade an der Überschrift arbeiten, machen wir einfach Letzteres.

```
#sidebar h4 {border: 1px solid rgb(73%,73%,58%);
  border-bottom: none;}
```

Nun werden wir als letzten Feinschliff den Hintergrund des Titels etwas dunkler färben. Das wird ihm einen subtilen visuellen Kick geben, der die anderen Teile des Layouts nicht unterdrückt.

```
#sidebar h4 {border: 1px solid rgb(73%,73%,58%);
  border-bottom: none;
  background: rgb(90%,90%,75%);}
```

Diese Änderung reicht, damit wir die zweite Hälfte des Projekts als vollendet erklären können, also genießen wir nun unser Werk. Das Endergebnis können Sie in Listing 5.4 und in der Abbildung 5.13 sehen.

Listing 5.4 Das Stylesheet für die »Links im Kasten«

```
html {margin: 0; padding: 0;}
body {font: 80% Verdana, Arial, Helvetica, sans-serif;
   margin: 0; padding: 0;
   background: rgb(95%,95%,80%); color: black;}
h1 {font-size: 200%; text-transform: lowercase; letter-spacing: 3px;
   margin: 0; padding: 0.66em 0 0.33em 29%;
   background: rgb(85%,85%,70%);}
h3 {font-size: 1.33em; margin: 0; padding: 0;
   border-bottom: 1px solid black;}
h4 {font-size: 1em; margin: 0; padding: 0.33em 0 0;
   color: rgb(50%,50%,35%);}
h1, h3, h4 {line-height: 1em;}
p {line-height: 1.5; margin: 0.5em 0 1em;}
div#entry {margin: 2em 10% 1em 30%; padding: 0;}
div#sidebar {float: left; width: 23%; margin: 2em 0 0 2%;
  text-align: center;}
#sidebar h4 {border: 1px solid rgb(73%,73%,58%);
  border-bottom: none;
  background: rgb(90%,90%,75%);}
#sidebar ul {list-style: none; margin: 0; padding: 0 0 2em;
  border: 1px solid rgb(73%,73%,58%);}
#sidebar h4, #sidebar ul {margin: 0 6px 0 0;}
#sidebar li {padding: 0.5em 0; line-height: 1em;
  border-bottom: 1px solid rgb(84%,84%,69%);}
#sidebar a {text-decoration: none;
  padding: 0 0.25em;
  border: 1px solid rgb(84%,84%,69%);
```

```
  background: rgb(95%,95%,80%);
  position: relative; top: 1em;}
#sidebar a:link {color: rgb(20%,40%,0%);}
#sidebar a:visited {color: rgb(58%,68%,40%);}
#sidebar a:hover {color: rgb(10%,20%,0%);
  background: #FFF;}
```

ABBILDUNG 5.13

Die Schlussarbeiten am Erscheinungsbild der Seitenleiste

Styler-Warnung

Erinnern Sie sich noch an die potenziellen Probleme mit dem Zeilenumbruch, die wir in der ersten Hälfte des Projekts besprochen haben? Sie betreffen alle Styles, die wir in der zweiten Hälfte geschrieben haben. Um zu sehen, was passieren kann, laden Sie sich die Datei `ch0513.html` *in einen Browser und machen das Browserfenster richtig schmal. Wenn die Zeilen umgebrochen werden, passiert das Gleiche mit den Rahmen, und das ist das korrekte Verhalten. Nervig vielleicht, aber korrekt.*

Das ist öfter der Fall bei CSS: Was für eine Situation passend ist, erscheint in einer anderen als ganz schlechte Wahl. Anders gesagt: Obwohl die in diesem Kapitel ausprobierten Techniken sehr nützlich sind, gilt das nicht für alle Situationen. Um CSS-Spezialist zu werden, muss man zum großen Teil einfach lernen, welche Techniken in welchen Situationen funktionieren, und dieses Wissen auf kreative Art anwenden.

Normalerweise sind die in diesem Projekt angewendeten Arten von Stylings am ehesten für Situationen geeignet, in denen kurze Elemente vorkommen, vorzugsweise solche mit einem einzelnen Wort, weil das nicht umgebrochen werden kann. Eine andere offensichtliche Situation ist, wenn Sie eine Liste mit grafischen Links (oder tatsächlich eine Liste mit beliebigen Bildern) stylen, weil Sie wissen, dass die Bilder nicht wortweise umgebrochen werden.

5.6 Spielwiese

Mit den uns zur Verfügung stehenden Elementen gibt es für das Styling unserer Menüs beinahe zu viele Auswahlmöglichkeiten. Hier sind noch ein paar weitere Vorschläge, für die Sie Ihr Gehirnschmalz nutzen können.

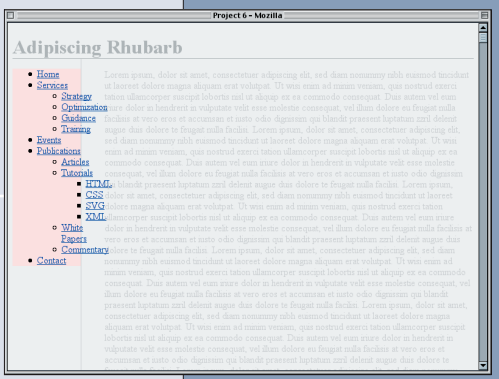
1. Ändern Sie den Rollover-Effekt aus der ersten Hälfte des Projekts so, dass er den gesamten Link mit einem Rahmen umschließt, ohne dass es beim Rollover einen Versatz nach links gibt. Das Hintergrundbild sollte sich auch auf die rechte Seite des Links bewegen. Da Sie gerade dabei sind, können Sie die Überschrift der Seitenleiste und die Überschrift des Haupttexts mittels negativer Ränder optisch verbinden, und fügen Sie Padding ein, damit der Text des Seitenleistentitels sich rechtsbündig mit dem doppelten Rahmen auf der rechten Seite der Seitenleiste ausrichtet.

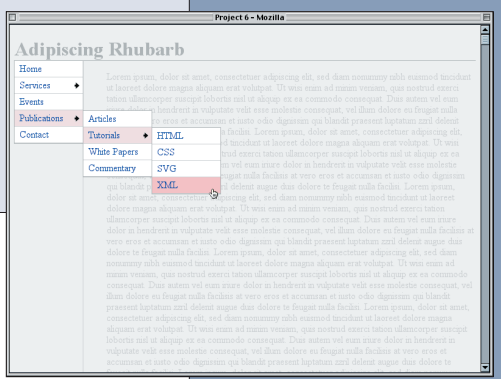
2. Schreiben Sie die Menü-Styles aus der zweiten Hälfte des Projekts um, so dass die Links so aussehen, als seien sie auf dem Hintergrund geschrieben, aber von einem dunklen Schatten umgeben sind, der dem im Seitenkopf verwendeten entspricht. Somit sollten die Links einen Hintergrund haben, der dem `body` entspricht, und sollten damit mit farbigen Linien, die sowohl zu den Links als auch zum `body` passen, »verbunden« sein.

3. Nehmen Sie die Styles aus der zweiten Hälfte des Projekts und kombinieren Sie sie mit Prinzipien aus der ersten Projekthälfte, um einen hybriden Style zu schaffen. Lassen Sie beispielsweise einen invertierten »L«-Rahmen um die obere und rechte Seite des Menüs laufen und lassen Sie jeden Link auf der linken Seite einer gedrehten »T«-Form sitzen, die aus dem »L« hervorgeht und genau rechts neben dem Text eines jeden Links endet.

6

Drop-down-Menüs mit CSS





Eltern, die eine neue Sicht auf ihre Möbel genießen wollen, sei geraten, sich auf alle Viere zu begeben und das neun oder zehn Monate alte Kind auf dessen Runden zu begleiten.

Selma H. Fraiberg

Wie wir im vorhergehenden Projekt gesehen haben, ist es möglich, eine einfache, unsortierte Liste mit Links zu nehmen und sie gut zu präsentieren. Uns fehlte als Komponente allerdings die Möglichkeit von Untermenüs oder gar Unter-Untermenüs. Gibt es nun einen Weg, wie man über CSS und einfaches HTML Drop-down-Menüs oder Menüs innerhalb anderer Menüs schaffen kann?

Wie Sie sicher schon gedacht haben, lautet die Antwort »Ja«. Hierbei müssen wir allerdings etwas proprietäre Technologie verwenden, um einen gewissen Browser mit ins Boot zu holen, aber weil es sich bei diesem Browser um den Internet Explorer für Windows handelt, ist der Nicht-Standard-Teil die Mühe wohl mehr als wert. Wie wir in diesem Projekt sehen werden, kann über einfache verschachtelte Linklisten eine Menüstruktur mit mehreren Ebenen erreicht werden.

6.1 Projektziele

Unser letztendliches Ziel ist einfach: Wir wollen eine Menüstruktur mit mehreren Ebenen schaffen und dafür nichts anders als CSS und ein paar unsortierte Listen nehmen. Diesem Gesamtziel ordnen sich einige konkretere Ziele unter:

- Die von uns erstellte Menüstruktur muss in der Lage sein, mehrere Ebenen von Drop-downs zu bewältigen, damit Menüs aus anderen Menüs heraus aufgerufen werden können.
- Diese Menüs sollen einfach entweder als Seitenleisten oder als horizontale Symbolleisten zu erstellen sein. Das soll auch möglich sein, ohne dass das Markup überhaupt geändert wird.
- Wenn wir JavaScript vermeiden können, werden wir das tun. Mit anderen Worten – wir wollen, dass die Menüs nur über CSS dynamisch gezeigt und verborgen werden. Dadurch verringert sich auch die Ladezeit der Seite, weil der Browser nur die Seite und das Stylesheet laden muss.
- Die Menüs müssen in so vielen Browsern wie möglich funktionieren – also auch insbesondere im Internet Explorer für Windows.

Das hört sich hier an, als läge ein Riesenberg Arbeit vor uns, aber der ganze Ablauf wird einfacher sein, als Sie annehmen. Der Löwenanteil unserer Arbeit besteht darin, die Menüs so anzulegen, wie wir sie haben wollen. Wenn das erst einmal erledigt ist, gibt es noch ein paar Kleinigkeiten, damit die Menüs dynamisch erscheinen und verschwinden, und dann können wir uns zurücklehnen und entspannen.

6.2 Vorbereitungen

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

Laden Sie die Dateien für Projekt 6 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch06proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und neu laden.

6.3 Die Grundlagen

Zuerst wollen wir herausfinden, wie das Dokument strukturiert ist. In diesem Fall gibt es im Dokument drei Hauptkomponenten:

- Der Titel der Seite
- Die Navigationslinks
- Der Hauptinhalt der Seite

Wie die Reihenfolge der Gliederungspunkte oben nahe legt, sind die Navigationslinks in einem `div` enthalten, das nach dem `h1` kommt, das den Seitentitel enthält, aber vor dem »Main«-`div`, das den Textinhalt der Seite beinhaltet. Die Datei verfügt bereits über ein paar grundlegende Styles, die Sie dem Listing 6.1 entnehmen und in Abbildung 6.1 anschauen können.

Listing 6.1 Die grundlegenden Styles

```
<style type="text/css">
body {background: #EEE; color: #000;}
h1 {color: #AAA; border-bottom: 1px solid; margin-bottom: 0;}
#main {color: #CCC; margin-left: 7em; padding: 1px 0 1px 5%;
  border-left: 1px solid;}
div#nav {float: left; width: 7em;
  background: #FDD;}
</style>
```

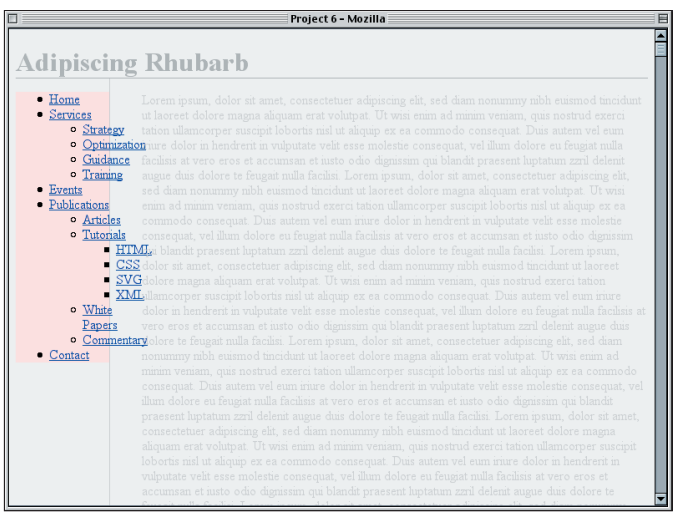


Abbildung 6.1

Unser Ausgangspunkt – die grundlegenden Styles sind bereits angewandt worden.

Diese grundlegenden Styles sind dazu da, die Seite korrekt anzuordnen und den Text optisch in der Betonung abzuschwächen, damit er uns nicht ablenkt, während wir an der Menüstruktur arbeiten. Der hellrote Hintergrund für das Menü-`div` ist einfach nur deswegen da, damit wir sehen können, wo das Navigations-`div` angelegt ist. Apropos Menüs – das Markup, über das sie erstellt werden, finden Sie in Listing 6.2.

Listing 6.2 Das Markup des Navigationsmenüs

```
<div id="nav">
<ul>
 <li><a href="/">Home</a></li>
 <li><a href="/services/">Services</a>
  <ul>
```

Benennung des `nav-div`

Sie fragen sich vielleicht, warum wir vom Navigations-`div` als `div#nav` statt einfach als `#nav` sprechen. Das tun wir, um einen Bug im Internet Explorer für Windows zu vermeiden: Er wird die Untermenüs nicht dynamisch anzeigen, wenn die Selektoren einfach `#nav` lauten. Wir wissen nicht genau, warum ein Hinzufügen des Elementnamens im Selektor diesen Bug repariert, aber es klappt.

```
    <li><a href="/services/strategy/">Strategy</a></li>
    <li><a href="/services/optimize/">Optimization</a></li>
    <li><a href="/services/guidance/">Guidance</a></li>
    <li><a href="/services/training/">Training</a></li>
   </ul>
  </li>
  <li><a href="/events/">Events</a></li>
  <li><a href="/pubs/">Publications</a>
   <ul>
    <li><a href="/pubs/articles/">Articles</a></li>
    <li><a href="/pubs/tuts/">Tutorials</a>
     <ul>
      <li><a href="/pubs/tuts/html/">HTML</a></li>
      <li><a href="/pubs/tuts/css/">CSS</a>
      <li><a href="/pubs/tuts/svg/">SVG</a>
      <li><a href="/pubs/tuts/xml/">XML</a>
     </ul>
    </li>
    <li><a href="/pubs/wpapers/">White Papers</a></li>
    <li><a href="/pubs/comment/">Commentary</a></li>
   </ul>
  </li>
  <li><a href="/contact/">Contact</a></li>
 </ul>
 </div>
```

Korrekte Verschachtelung

Beachten Sie, dass jede verschachtelte Liste sich in einem Listenelement aus der übergeordneten Liste befindet – beispielsweise enthält das Listenelement für *Services* einen Link, eine verschachtelte Liste, und dann erscheint das `</li>`. Das ist korrekt, weil Listen keine unmittelbar untergeordneten Elemente von Listen, sondern nur von Listenelementen sein können. Ein weitverbreiteter Fehler besteht darin, dass eine verschachtelte Liste *zwischen* den Listenelementen platziert wird statt innerhalb.

Ganz genau – es sind nur ein paar verschachtelte, unsortierte Listen, in denen Links enthalten sind, und mehr nicht. Das ist weitgehend alles, was wir brauchen. Im weiteren Verlauf des Projekts werden wir dem Markup ein paar Klassen zuweisen, damit die Menüstruktur funktioniert, aber das ist auch schon alles.

6.4 Layout für die Menüs

Bevor wir an den Punkt kommen, die Untermenüs auf- und zuklappen zu lassen, werden wir sie erst einmal stylen, damit sie auch anständig aussehen. Anders gesagt – wir werden alle Menüs und Untermenüs stylen und erst dann die nötigen Maßnahmen ergreifen, um sie entsprechend der Aktionen des Users erscheinen und verschwinden zu lassen.

6.4.1 Vorausplanung

Unser erster Schritt wird sein, eine Behavior-Datei aufzurufen, mit der wir die nur in Windows Internet Explorer enthaltene Eigenschaft `behavior` verwenden. Wir werden also eine separate Datei einbringen, die dem IE/Win Fähigkeiten verleiht, die er normalerweise nicht hat.

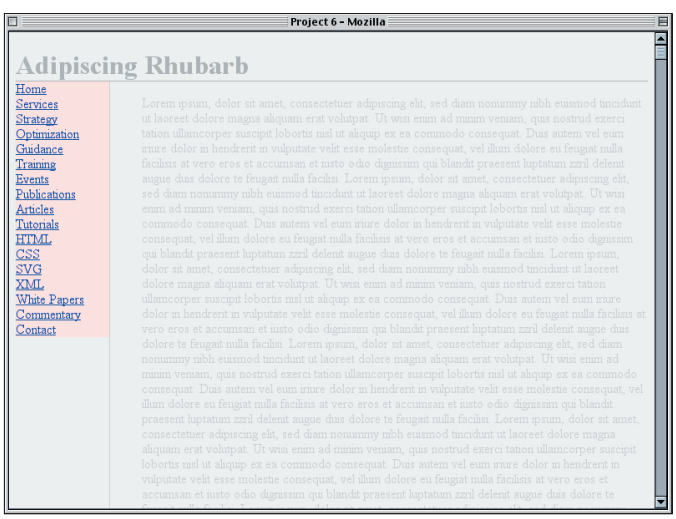
```
body {background: #EEE; color: #000;
  behavior: url(csshover.htc);} /* Aufruf einer WinIE-Behavior-Datei */
```

Damit das funktioniert, brauchen Sie die `.htc`-Datei von der Website dieses Buchs oder eine Kopie von ihrem Autor Peter Nederlof. Seine Demonstration dieser Technik, bei der er die gleiche `.htc`-Datei benutzt wie wir hier, können Sie unter `http://www.xs4all.nl/~peterned/csshover.html` finden.

Bevor Peter seine `csshover.htc` veröffentlicht hatte, konnte IE/Win nur dann Systeme mit CSS-Menüs unterstützen, wenn der Autor dem IE/Win mit etwas JavaScript auf die Sprünge geholfen hatte. Peter hat gewissermaßen das Gleiche gemacht, außer dass er diese zusätzlichen Fähigkeiten in eine Windows-Behavior-Datei gepackt hat anstatt in eine JavaScript-Datei. Das hat einen Riesenvorteil: Nur der IE/Win wird diese HTC-Datei herunterladen, und das heißt, andere Browser (die das sowieso nicht brauchen) werden keine Bandbreite verschwenden, um diese Datei zu holen. Bei einer Lösung über JavaScript müssen alle Browser das Skript herunterladen.

Die Herangehensweise über das Verhalten birgt nur einen Nachteil: Die Deklaration `behavior` verhindert, dass das Stylesheet validiert wird. Wenn das für Sie besonders wichtig ist, könnten Sie die `behavior`-Deklaration in ein separates Stylesheet verschieben und sie dann über `@import` einbinden. Darüber könnte das Haupt-Stylesheet validiert und die nicht-valide `behavior`-Deklaration in eine Extra-Datei ausgelagert und ignoriert werden. Für was Sie sich entscheiden, hängt wohl davon ab, wie Sie Validierung betrachten.

Momentan ändert die Behavior-Datei überhaupt nichts an unserer Seite, vor allem, da wir noch gar nicht an dem Punkt angekommen sind, wo sie nützlich wäre. Zuerst wollen wir nun unsere Menüs stylen. Als ersten Schritt werden wir alle Einrückungen hinauswerfen und die Breite der Liste so einstellen, dass sie genauso groß ist wie die Breite des `div`s. Das Ergebnis sehen Sie in Abbildung 6.2.



Ränder und Padding

Die Einzüge von Listen werden in einigen Browser über Ränder und in anderen über das Padding bewirkt. Stellen wir beides auf Null, gehen wir absolut auf Nummer sicher.

ABBILDUNG 6.2

Nach der Entfernung der Einrückung von den Listen passen diese viel besser in die Seitenleiste.

```
div#nav {float: left; width: 7em;
  background: #FDD;}
div#nav ul {margin: 0; padding: 0; width: 7em;}
</style>
```

Momentan ist es viel schwerer geworden zu erkennen, wo die Untermenüs anfangen und aufhören. Darum werden wir uns in der nächsten Phase des Projekts in einem Rundumschlag kümmern.

6.4.2 Positionierung der Untermenüs

Als nächsten Schritt werden wir die Untermenüs dort platzieren, wo sie erscheinen sollen, wenn das System vollständig funktioniert. Eine sehr verbreitete Technik ist, dass ein Untermenü oben ausgerichtet und rechts von dem Element erscheint, das sein Erscheinen ausgelöst hat. Das können wir sehr leicht über eine Positionierung der Listen unter Beachtung der Listenelemente, in denen sie sich befinden, erzielen.

Dafür müssen wir zuerst gewährleisten, dass alle Listenelemente als Ausgangspunkt für die Position des jeweiligen untergeordneten Untermenüs agieren, indem wir einen *Container-Block* einrichten. Das ist so einfach wie das relative Positionieren der Listenelementen ohne Offsets.

```
div#nav ul {margin: 0; padding: 0; width: 7em;}
div#nav li {position: relative;}
</style>
```

Dadurch werden alle Listenelemente einen Container-Block (betrachten Sie ihn als einen Positionierungskontext) für alle Nachfahr-Elemente einrichten, die absolut positioniert sind. Wenn wir also die Links innerhalb der Listenelemente über `right:0;` absolut positioniert hätten, würde jeder Link genau neben der rechten Kante des Listenelements, in dem er enthalten ist, positioniert.

Natürlich werden wir keine Links positionieren, sondern unsortierte Listen. Alle Listen, die sich in einer anderen Liste befinden, werden wir so platzieren, dass die Oberkante des Untermenüs auf gleicher Höhe mit der Oberkante des Listenelements ist.

```
div#nav li {position: relative; list-style: none; margin: 0;}
div#nav ul ul {position: absolute; top: 0;}
</style>
```

Somit wird die obere Kante aller Untermenüs am oberen Rand seines Container-Blocks entlang ausgerichtet. Jetzt brauchen wir das Untermenü nur noch genau neben das Listenelement stellen, in dem es enthalten ist. Dafür werden wir nun ein bisschen zaubern. Dank unserer Regel `div#nav ul` wissen wir, dass die Listen alle `7em` breit sind. Das heißt, dass die rechten Kanten der Listenelemente alle `7em` rechts neben ihren linken Kanten sind. Wenn wir die positionierten Untermenüs um den gleichen Betrag verschieben, werden sie genau da platziert, wo wir sie haben wollen – schauen Sie sich Abbildung 6.3 an.

```
div#nav ul ul {position: absolute; top: 0; left: 7em;}
```

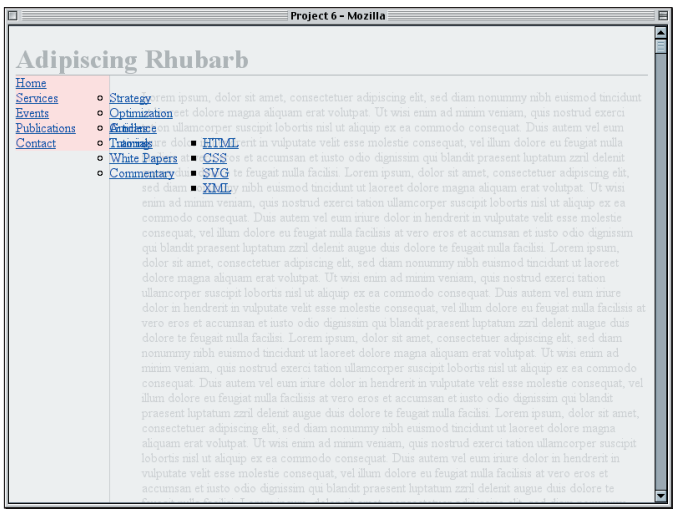


ABBILDUNG 6.3

Die Untermenülisten werden bezogen auf die in ihnen enthaltenen Listenelemente positioniert.

Bei der Abbildung 6.3 gibt es vier Dinge anzumerken:

- Das Navigations-`div` bleibt genauso groß wie sein Inhalt. Jetzt, wo wir die Untermenüs absolut positioniert haben, sind sie eigentlich nicht mehr länger Teil seines Inhalts – jedenfalls nicht vom Standpunkt des Layouts betrachtet.
- Die letzten beiden Elemente im ersten Untermenü überlappen die ersten beiden aus dem zweiten Untermenü. Damit sind sie momentan schwer zu lesen, aber Überlappungen stellen im Prinzip kein großes Problem dar, denn wenn wir fertig sind, wird von diesen Untermenüs immer nur eines zur Zeit erscheinen.
- Die erste Ebene der Untermenüs ist genau an der rechten Kante der Links der Hauptebene ausgerichtet – (*Home*, *Services* usw.). Die Gliederungspunkte (*Bullets*), die den Listenelementen sozusagen angefügt sind, ragen tatsächlich aus den unsortierten Listen heraus. Wir können sie bei den Haupt-Links nicht sehen, weil die Bullets in Wirklichkeit links von der linken Kante des Browserfensters erscheinen.
- Und die Listen haben immer noch Gliederungspunkte, wenigstens in den meisten Browsern. Die sollten wir mal schnell loswerden.

Bevor wir weitermachen, tragen wir also erst einmal dies ein:

```
div#nav li {position: relative; list-style: none; margin: 0;}
```

Damit fliegen die Bullets raus, wie wir in der nächsten Abbildung sehen können.

6.4.3 Schönere Menü-Styles

Nun sind die Untermenüs dort platziert, wo wir sie haben wollen, also ist es Zeit, ihr Äußeres aufzupolieren. Sie sollen einen weißen Hintergrund und – zumindest für den Augenblick – einen soliden Rahmen von einem Pixel bekommen, dessen Farbe dem Wert `color` der `ul`-Elemente entnommen wurde.

```
div#nav ul {margin: 0; padding: 0; width: 7em;
  background: white; border: 1px solid;}
```

Damit werden alle unsortierten Listen – vom obersten Level bis zum tiefsten Untermenü – in einen Kasten gelegt. Weil einige Browser wie z.B. Opera bei Listenelementen standardmäßig Ränder anwenden, werden wir auch ihre Ränder auf Null stellen.

```
div#nav li {position: relative; list-style: none; margin: 0;}
```

Benennung von Klassen

Statt `submenu` können Sie auch praktisch jeden anderen Namen verwenden. Wir hätten auch `sm`, `subm` oder sogar `claudia` nehmen können. Wir haben uns für den Wert `submenu` entschieden, weil er unmissverständlich ist. Wenn Sie an Ihren eigenen Designs arbeiten, können Sie sich gerne einen Namen aussuchen, der Ihnen mehr zusagt.

Jetzt zu den Untermenüs: Damit das Styling später leichter wird, werden wir dem Markup einige strukturelle Hooks beigeben. Für jedes der drei Listenelemente, die ein Untermenü enthalten, werden wir ein `class`-Attribut mit dem Wert `submenu` hinzufügen, wie Sie hier sehen können:

```
<li class="submenu"><a href="/services/">Services</a>
<li class="submenu"><a href="/pubs/">Publications</a>
   <li class="submenu"><a href="/pubs/tuts/">Tutorials</a>
```

In Wirklichkeit markieren wir die Listenelemente, die ein Untermenü enthalten, mit dem Etikett `submenu`. Um sicherzugehen, dass wir die richtigen Elemente klassifiziert haben, werden wir eine Regel zur Hervorhebung einfügen – und das Ergebnis sehen Sie in Abbildung 6.4.

```
div#nav li {position: relative; list-style: none; margin: 0;}
div#nav li.submenu {background: yellow;}
div#nav ul ul {position: absolute; top: 0; left: 7em;}
```

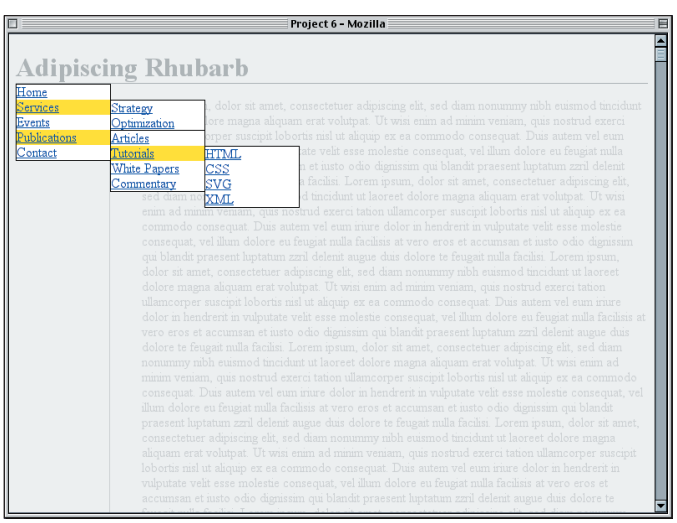


ABBILDUNG 6.4

Das Erscheinungsbild der Untermenüs ist verbessert worden und die Listenelemente, in denen Untermenüs enthalten sind, werden hervorgehoben.

Jetzt, wo alle Menüs Hintergründe haben, können Sie die Überlappung der ersten beiden Untermenüs aus der obersten Ebene leichter verstehen. Sie überlappen immer noch, aber der Hintergrund des zweiten Untermenüs verdeckt das erste komplett. Und bei dem Unter-Untermenü können wir sehen, dass seine linke Kante nun schön mit der rechten Kante des übergeordneten Listenelements abschließt.

6.4.4 Beschränkte Links

Wenn Sie dem Projekt mit Ihren eigenen Dateien folgen, ist Ihnen an den letzten Änderungen vielleicht etwas aufgefallen. Der anklickbare Bereich der Links in den Menüs füllt nicht das ganze Listenelement aus. Das liegt daran, dass die Links Inline-Elemente sind. Wenn wir den Links beispielsweise einen Rahmen geben, würde der Rahmen nur den Text einfassen.

Damit die Menüs korrekt arbeiten und sich wie von den Usern erwartet verhalten, müssen wir dafür sorgen, dass die Links wirklich die gesamten Listenelemente ausfüllen. Dafür werden wir die Art der Boxen ändern, die die Links bei ihrer eigenen Erstellung anlegen.

```
div#nav li.submenu {background: yellow;}
div#nav li a {display: block;}
div#nav ul ul {position: absolute; top: 0; left: 7em;}
```

Jetzt wird jeder Link eine Blocklevel-Box generieren, so ähnlich wie es auch über einen `div` oder einen Absatz geschieht. Beachten Sie, dass wir die Art der Links selbst *nicht* geändert haben. Die `a`-Elemente sind immer noch Inline-Elemente. Hier sorgt CSS nur dafür, dass sie Blockboxen generieren. Das ist nur ein kleiner, aber feiner Unterschied, den man sich merken sollte.

Jetzt können wir nicht nur innerhalb eines Menüeintrags irgendwo hinklicken und der funktioniert dann auch (außer in IE/Win, aber das reparieren wir gleich), sondern wir können die Links auch so stylen wie ein Blocklevel-Element. Um den Link-Text beispielsweise ein wenig nach rechts zu schieben, geben wir den Links etwas Padding auf der linken Seite. Wir können die Links auch mit etwas oberem und unterem Padding auseinander schieben.

```
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;}
```

Die Unterstreichung der Links sollten auch ausgeschaltet werden, da User nicht erwarten, dass Menü-Links unterstrichen sind.

```
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;
  text-decoration: none;}
```

Jetzt wenden wir uns den Problemen des Internet Explorers für Windows zu. Beim IE/Win füllen Blockbox-Links wie unsere aus was für Gründen auch immer nicht das gesamte Listenelement aus. Das ist ein Bug, aber glücklicherweise einer, für den es ein Workaround gibt. Wenn wir den Links eine explizite Breite geben, bekommen wir das

Verdoppelte Höhen

Im IE/Win werden die Listen über den eben gerade eingefügten `display: block;` auseinander gezogen. Das liegt an einem Bug im Umgang des Internet Explorers für Windows mit Whitespace und Blockbox-Links in Listen. Aber keine Sorge: Wir werden den Bug außer Gefecht setzen, bevor wir zur nächsten Abbildung kommen.

gewünschte Verhalten. Andererseits soll bei anderen Browsern wie vorher auch schon eine automatische Breite eingestellt werden.

Dafür müssen wir zuerst explizit eine Breite für die Links angeben und dann eine Regel festlegen, die IE/Win nicht versteht, um diese Breitenangabe wieder auszuhebeln.

```
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;
  text-decoration: none; width: 6.5em;}
div#nav>ul a {width: auto;}
div#nav ul ul {position: absolute; top: 0; left: 7em;}
```

Mit einer Content-Breite von `6.5em` und einem linken Padding von `0.5em` wird die Elementbox des Links genau wie das übergeordnete Element 7em breit sein. Jetzt können wir einfach in allen Listenelementen unten einen hellgrauen Rahmen einfügen und sie damit optisch voneinander trennen.

```
div#nav li {position: relative; list-style: none; margin: 0;
  border-bottom: 1px solid #CCC;}
```

Wir hätten den Rahmen auch unten bei den Links einfügen können. So, wie wir die Menüs gestylt haben, hätte das genauso funktioniert wie das Stylen der Listenelemente. Das Ergebnis all dieser Änderungen zusammengenommen sehen Sie in Abbildung 6.5.

Wie war das doch gleich?

Der Selektor `div#nav>ul a` besagt: »Jedes `a`-Element, das Nachkömmling von einem `ul`-Element ist, das selbst ein unmittelbar untergeordnetes Element (»Kind«) eines `divs` mit der `id nav` ist«. Damit werden alle Links in unserer Menüstruktur ausgewählt. IE/Win versteht den Selektor-Teil mit »Kindelement eines `divs`« nicht, überspringt also die gesamte Regel.

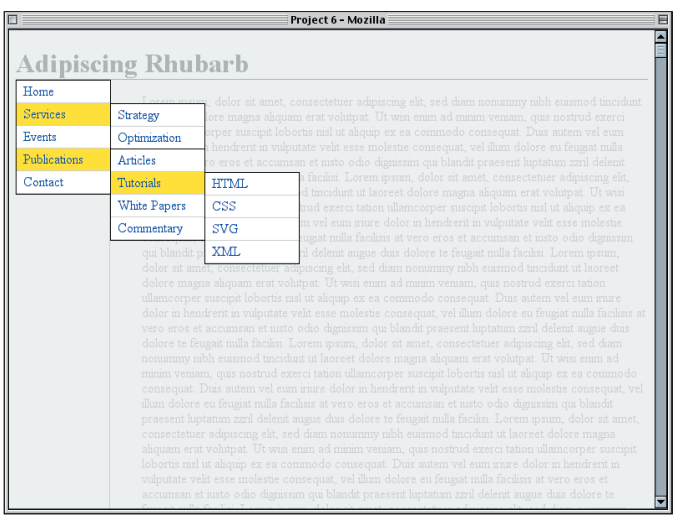


ABBILDUNG 6.5

Padding der Links verteilt die Listenelemente und die Links füllen ihre Listenelemente ganz aus.

Box-Modell-Blues

Die Deklaration einer `width` *von* `6.5em` *für die Links funktioniert prima im IE6, der mit dem CSS-Box-Modell gut zurecht kommt. Bei älteren Versionen des IE/Win ist das leider nicht der Fall, weil diese das Modell falsch verstehen. Man kann eine Reihe von Regeln konstruieren, die all das berücksichtigen. Dazu gehört die Verwendung des »Box-Modell-Hacks« und auch des Kind-Selektor-Hacks, den wir gerade gesehen haben. Ein Beispiel:*

```
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;
  text-decoration: none; width: 100%;
  voice-family: „\"}\"“; voice-family:inherit;
  width: 6.5em;}
div#nav>ul a {width: auto;}
div#nav ul ul {position: absolute; top: 0; left: 7em;}
```

Die erste Änderung stellt die Linkbreite auf `100%` *ein, was bei älteren Versionen des IE/Win erforderlich ist. Das ist der letzte* `width`*-Wert, der von diesen älteren Versionen erkannt wird, weil die nächste Zeile (mit den* `voice-family`*-Deklarationen) sie dazu bringt, den Rest der Regel zu überspringen. IE6 wird trotzdem weitermachen und auf die Deklaration* `width: 6.5em;` *treffen. Das Gleiche gilt für Opera, Mozilla, Safari und andere moderne Browser. Die Zeile mit dem Kind-Selektor, bei der die* `width` *wieder auf* `auto` *zurückgestellt wird, wird dann vom IE6 und älteren Versionen des IE/Win ignoriert, aber von modernen Browsern verarbeitet.*

Ob dieser ganze Aufwand den Effekt rechtfertigt, hängt vor allem von der demographischen Struktur Ihrer Zielgruppe ab. Wenn Sie viele User mit älteren Rechnern haben, werden Sie sich wohl diese Mühe machen. Für den Rest des Projekts werden wir davon ausgehen, dass unsere Zielgruppe die neuesten Versionen ihres jeweiligen Browsers verwendet, und lassen die komplizierteren Serien von Hacks wie die gerade gezeigte einfach aus.

6.4.5 Rollover-Einträge

Es wird nun Zeit, den Rollover-Style bei den Menüeinträgen vorzunehmen, damit der Hintergrund die Farbe ändert, wenn der User den Mauszeiger über »Home« hält. Das kriegen wir mit der folgenden Regel hin:

```
div#nav li {position: relative; list-style: none; margin: 0;
  border-bottom: 1px solid #CCC;}
div#nav li:hover {background: #EBB;}
div#nav li.submenu {background: yellow;}
```

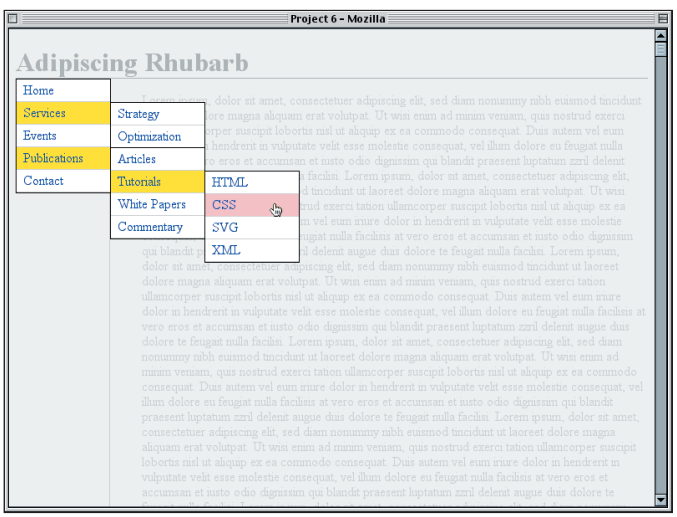
Nein, das ist kein Fehler, also bitte nicht gleich Korrektur-E-Mails schicken. Wir wenden mit voller Absicht einen Rollover-Effekt auf die Listenelemente selbst und nicht auf die Links an. Das wird von CSS vollkommen akzeptiert, weil Rollover-Effekte

nicht nur auf Hyperlinks beschränkt sind: Jedem beliebigen Element kann ein Rollover-Style zugewiesen werden. Also wird sich in diesem Fall der Hintergrund aller `li`-Elemente, über denen der Mauszeiger innehält und die einem `div` mit einer `id` von `nav` nachfolgen, zu `#EBB` ändern.

Warum setzen nicht mehr Autoren so etwas ein? Weil der IE/Win tatsächlich die Rollover-Effekte auf Hyperlinks beschränkt und sie auf keine anderen Elementarten anwendet. Wenigstens in der Standardeinstellung passiert das nicht. Erinnern Sie sich noch an die Datei `csshover.htc`, die wir vorhin am Anfang dem Element `body` zugeordnet haben? Der einzige Zweck dieser Datei ist, im IE/Win beliebigen Elementen Rollover-Fähigkeiten zu verleihen. Anders gesagt verdanken wir es dieser Behavior-Datei, dass der IE/Win nun Rollover-Effekte bei allen Elementen erlaubt – so wie in Abbildung 6.6 zu sehen ist.

ABBILDUNG 6.6

Listenelemente erhalten Rollover-Effekte.



Wiederum wechselt der Hintergrund bei dem *Listenelement*, nicht dem Link innerhalb des Listenelements. Diese Links haben alle transparente Hintergründe, so dass wir sehen können, wie die Hintergründe der Listenelemente hinter ihnen wechseln.

Die Tragweite dieser Fähigkeit ist nicht leicht zu verstehen. Zusätzlich zur Möglichkeit für verschachtelte Pop-up-Menüs, die auf einfachem Markup und CSS basieren, kann diese Fähigkeit dazu genutzt werden, dass sich Informationen in einem Bereich der Seite ändern, wenn der Mauszeiger über einem anderen Bereich ruht – und noch eine ganze Menge mehr. Der Einsatz von ein wenig nicht-validem CSS scheint sich zu lohnen, wenn man dafür beim IE/Win Rollover-Effekte für beliebige Elemente bekommen kann.

6.4.6 Spezielle Styles für Untermenüs

Atmen Sie einmal tief durch, und dann können wir weitermachen. Die gelbe Hervorhebung bei den Untermenü-Links hat seinem Zweck gedient, aber sie geht uns langsam auf die Nerven. Wir wollen immer noch einen optischen Hinweis dafür, welche Menü-Einträge ein weiteres Menü öffnen werden, also werden wir das Gelb durch ein Hintergrund-Bild mit Pfeil namens `submenu.gif` ersetzen.

```
div#nav li.submenu {background: url(submenu.gif) 95% 50% no-repeat;}
```

Dies wird eine einzelne Instanz des Bildes vertikal zentriert rechts platzieren, und zwar 95 % von der Entfernung zwischen den Untermenü-Listenelementen (siehe Abbildung 6.7).

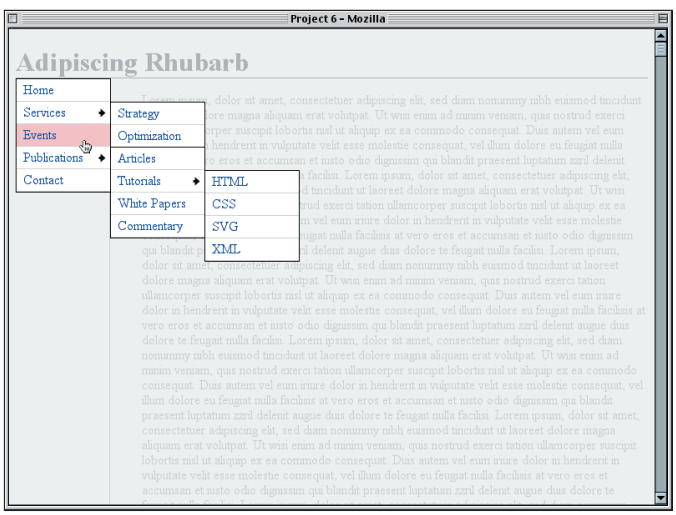


ABBILDUNG 6.7

Die Platzierung eines Pfeils zeigt an, hinter welchen Einträgen sich Untermenüs verbergen.

Das sieht alles schon viel besser aus. Allerdings heißt das nun wiederum, dass der gleiche Rollover-Effekt auf jedes Listenelement angewendet wird: Sie werden alle eingefärbt – egal ob ein Untermenü vorhanden ist oder nicht. Wir wollen nun die Rollover-Farbe für die Untermenü-Einträge aufhellen. Wieder werden wir die Rollover-Styles nicht auf die Links, sondern direkt auf die Listenelemente anwenden.

```
div#nav li.submenu {background: url(submenu.gif) 95% 50% no-repeat;}
div#nav li.submenu:hover {background-color: #EDD;}
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;
  text-decoration: none; width: 6.5em;}
```

Wir könnten die Links natürlich auch direkt stylen, aber wenn wir das für die Untermenü-Einträge machen, würde der Hintergrund des Links die Pfeil-Grafik für den Hintergrund des Listenelements verbergen. Wenn wir das gewollt hätten, wäre es eine prima Lösung, aber so soll es nicht sein. Indem wir die Hintergrundfarbe des Listenelements ändern, können Bild und Farbe gemeinsam existieren (siehe Abbildung 6.8).

Seltsamer Explorer

Wenn Sie IE/Win verwenden, fallen Ihnen vielleicht ein paar Merkwürdigkeiten bei den Rollover-Effekten auf, beispielsweise dass das Pfeil-Bild verschwindet. Das liegt wohl an einem fehlerhaften Behavior-Skript oder im Umgang des Explorers damit. Diese Sonderbarkeiten sind so gering, dass man sie ignorieren kann, aber behalten Sie die Mängel des Skripts im Hinterkopf, wenn Sie es für ein öffentliches Design verwenden wollen.

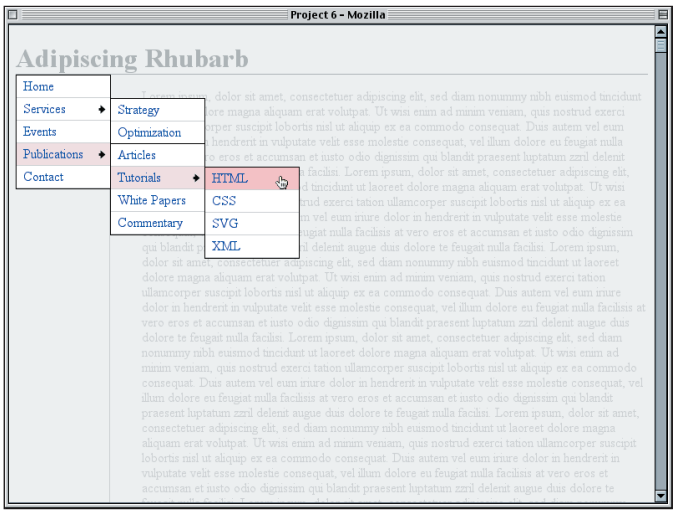


ABBILDUNG 6.8
Untermenü-Einträge erhalten einen anderen Rollover-Style.

6.4.7 Pop-ups tauchen auf

Mittlerweile sind wir nun soweit, dass wir alle Styles, die wir für die Präsentation und Platzierung der Menüs, Untermenüs und individuellen Einträge brauchen, fertig haben. Nun werden wir uns also noch um den Prozess des Anzeigens und Verbergens kümmern.

Bevor wir mehr CSS einfügen, wollen wir uns ein paar Änderungen für das HTML selbst anschauen. Wir werden den verschiedenen Listen Klassen zuordnen, damit wir das Ein- und Ausklappen der Untermenüs wirklich über das Stylesheet kontrollieren können. Diese Änderungen sind im Listing 6.3 hervorgehoben.

Listing 6.3 Hinzugefügte Level-Infos

```
<div id="nav">
<ul class="level1">
 <li><a href="/">Home</a></li>
 <li class="submenu"><a href="/services/">Services</a>
  <ul class="level2">
   <li><a href="/services/strategy/">Strategy</a></li>
   <li><a href="/services/optimize/">Optimization</a></li>
   <li><a href="/services/guidance/">Guidance</a></li>
   <li><a href="/services/training/">Training</a></li>
  </ul>
 </li>
 <li><a href="/events/">Events</a></li>
 <li class="submenu"><a href="/pubs/">Publications</a>
  <ul class="level2">
   <li><a href="/pubs/articles/">Articles</a></li>
   <li class="submenu"><a href="/pubs/tuts/">Tutorials</a>
```

```
      <ul class="level3">
       <li><a href="/pubs/tuts/html/">HTML</a></li>
       <li><a href="/pubs/tuts/css/">CSS</a>
       <li><a href="/pubs/tuts/svg/">SVG</a>
       <li><a href="/pubs/tuts/xml/">XML</a>
      </ul>
     </li>
     <li><a href="/pubs/wpapers/">White Papers</a></li>
     <li><a href="/pubs/comment/">Commentary</a></li>
    </ul>
   </li>
   <li><a href="/contact/">Contact</a></li>
  </ul>
  </div>
```

Und warum brauchen wir diese Änderungen? Wenn wir in unserem CSS einfach geschrieben hätten: »Zeig jedes `ul`, das Nachfahre eines Listenelements mit Rollover-Effekt aus einem Untermenü ist«, dann würde, wenn der Mauszeiger über *Publications* steht, sofort sowohl das Untermenü für *Publications* als auch das von *Tutorials* erscheinen. Stattdessen sollten wir lieber sagen: »Zeige jedes `ul`, das ein Kind-Element eines Untermenü-Listenelements mit Rollover-Effekt ist«. Und das kann über einen Kind-Selektor erreicht werden, den der IE/Win nicht versteht. Als Workaround dafür müssen wir die Menüs ihrem Verschachtelungslevel entsprechend klassifizieren.

Weil unsere Untermenüs bereits sichtbar sind, müssen wir sie verstecken. Da wir jedes `ul` verstecken, das Nachfahre eines `ul` ist, können wir eine bereits existierende Regel modifizieren, und deren Ergebnis ist in Abbildung 6.9 zu sehen.

```
div#nav ul ul {position: absolute; top: 0; left: 7em;
  display: none;}
```

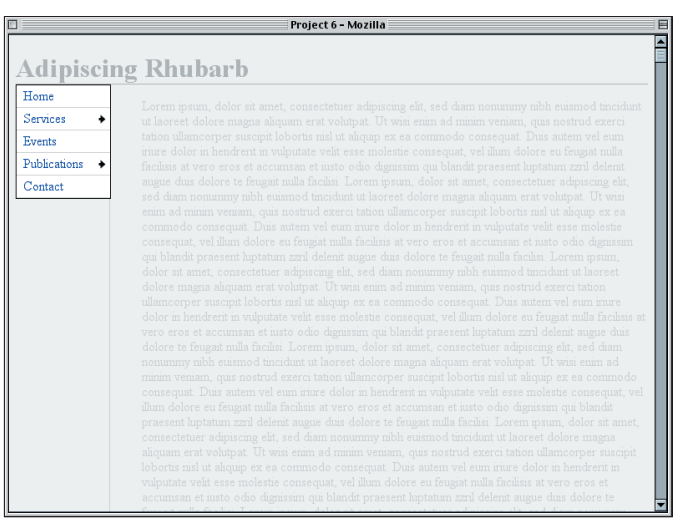


ABBILDUNG 6.9
Versteckte Untermenüs

Als Standard sehen wir also nur die oberste Ebene des Menüs. Das ist gut! Genau, wie wir es haben wollten.

Und jetzt stellen wir das Erscheinen der Untermenüs ein. Es soll jedes Mal, wenn der Mauszeiger über dem übergeordneten Listenelement steht, ein Menü der zweiten Ebene erscheinen, und wir bringen es wieder zum Vorschein, indem wir sein `display` auf `block` stellen.

```
div#nav ul ul {position: absolute; top: 0; left: 7em;
  display: none;}
div#nav ul.level1 li.submenu:hover ul.level2 {display:block;}
</style>
```

So wird jede `level2`-Liste ihr `display` von `none` zu `block` ändern, wenn sie von einem Listenelement mit Rollover-Effekt abstammt, das selbst von einer `level1`-Liste abstammt (und diese wiederum von einem `div` mit dem Individualformat `nav` abstammt). Das klappt mit Menüs der zweiten Ebene ganz prima, aber mit der dritten Ebene? Da brauchen wir bloß den Selektor zu ändern.

```
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
```

Jetzt wird die Regel auch bei Menüs der dritten Ebene genau wie bei denen der zweiten Ebene angewendet, und das Ergebnis sehen Sie in Abbildung 6.10.

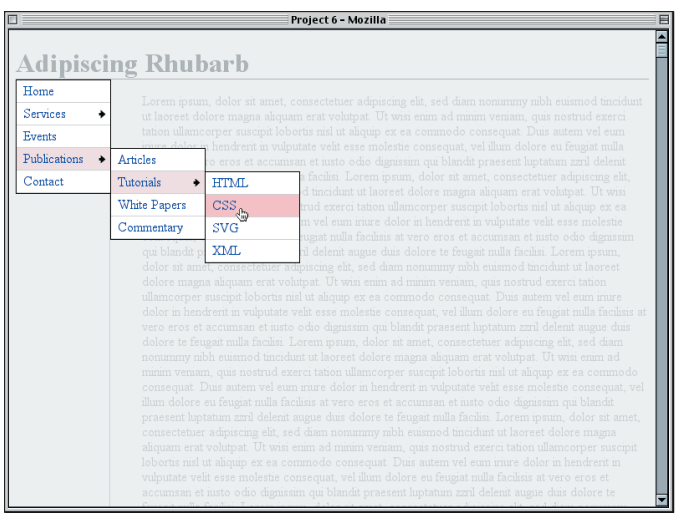


ABBILDUNG 6.10

Die Untermenüs werden den User-Aktionen entsprechend angezeigt.

Und mehr brauchen Sie wirklich nicht, um mit CSS und ein paar Behavior-Scripts verschachtelte Pop-up-Menüs zu erstellen.

display oder visibility?

Statt `display: none` *und* `display: block` *zu verwenden, um die Menüs anzuzeigen und zu verbergen, könnten wir auch* `visibility` *einsetzen. In diesem Fall lauten die relevanten Regeln im Stylesheet folgendermaßen:*

```
div#nav ul ul {position: absolute; top: 0; left: 7em;
  visibility: hidden;}
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {visibility: visible;}
```

Der Unterschied besteht darin, dass ein Element mit `display: none` *überhaupt keine Elementbox generiert; es ist, als existierte das Element überhaupt nicht. Ein Element, bei dem* `visibility: hidden` *eingestellt ist, generiert immer noch eine Elementbox mit der gleichen Größe, die es hätte, wenn sie sichtbar wäre, aber die Elementbox ist komplett transparent.*

In einem Layout mit normalem Fluss macht das schon einen Unterschied, weil eine transparente (versteckte) Elementbox das Layout für andere Elemente beeinflusst. Weil die Menüs absolut positioniert sind, wirken sie sich jedoch nicht auf das Layout anderer Elemente aus. Darum funktionieren `visibility` *und* `display` *in dieser Situation beide gleich gut. Wir bleiben bei* `display`*, weil Peter das in seiner ursprünglichen Demo verwendet hat und es ganz prima funktioniert.*

6.4.8 Der letzte Feinschliff

Nun, da unser Menüsystem funktioniert, wollen wir der Präsentation noch eine besondere Note geben. Wie wir aus der vorhergehenden Abbildung sehen können, stechen die Menüs durch ihre schwarzen Rahmen recht krass hervor, und die oberen und rechten Rahmen der Menüs der Hauptebene setzen neben den Rahmen für den Haupttext und den Seitentitel an und verdoppeln diese.

Das erste Problem beheben wir, indem wir das Menü mit Hilfe von negativen Rändern um einen Pixel nach oben und links schieben.

```
div#nav {float: left; width: 7em; margin: -1px 0 0 -1px;
  background: #FDD;}
```

Die Rahmen der Liste können abgeschwächt werden, indem sie passend zu der bereits verwendeten Farbe für den Titel gemacht werden. Dies wird in Abbildung 6.11 gezeigt.

```
div#nav ul {margin: 0; padding: 0; width: 7em; background: white;
  border: 1px solid #AAA;}
```

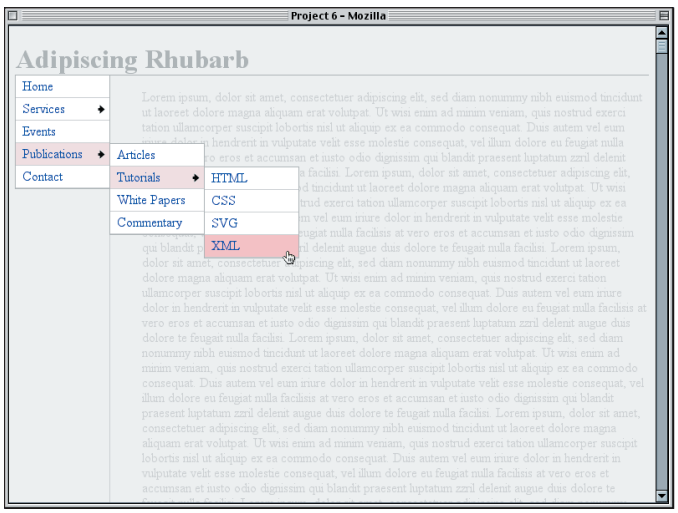
Jetzt passt alles schön zusammen, und das verdanken wir nur dem Stylesheet (plus der Behavior-Datei, deren Listing hier nicht erscheint), das Sie in Listing 6.4 finden.

Beschnitt durch negative Ränder

Bei den Tests fiel auf, dass einige Versionen des IE/Win die negativen oberen und linken Ränder das Menü nach links und oben schieben, aber gleichzeitig die Rahmen an diesen Seiten abschneiden. Wenn Sie das nicht sehen können, sollten Sie es sich trotzdem merken, dass es passieren kann und dass man das vor einer Veröffentlichung des Layouts berücksichtigen sollte.

ABBILDUNG 6.11

Das Erscheinungsbild des Menüs wird dem Rest des Layouts mehr angeglichen.



Listing 6.4 Die Styles des Menüs

```
body {background: #EEE; color: #000;
  behavior: url(csshover.htc);} /* WinIE behavior call */
h1 {color: #AAA; border-bottom: 1px solid; margin-bottom: 0;}
#main {color: #CCC; margin-left: 7em; padding: 1px 0 1px 5%;
  border-left: 1px solid;}
div#nav {float: left; width: 7em; margin: -1px 0 0 -1px;
  background: #FDD;}
div#nav ul {margin: 0; padding: 0; width: 7em; background: white;
  border: 1px solid #AAA;}
div#nav li {position: relative; list-style: none; margin: 0;
  border-bottom: 1px solid #CCC;}
div#nav li:hover {background: #EBB;}
div#nav li.submenu {background: url(submenu.gif) 95% 50% no-repeat;}
div#nav li.submenu:hover {background-color: #EDD;}
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;
  text-decoration: none; width: 6.5em;}
div#nav>ul a {width: auto;}
div#nav ul ul {position: absolute; top: 0; left: 7em;
  display: none;}
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
```

6.5 Neuausrichtung der Menüs

So weit, so gut – wir haben eine gute Menüstruktur mit mehreren Ebenen kreiert, in dem jedes Untermenü sich rechts von seinem übergeordneten Menüpunkt öffnet. Das ist eine schöne Sache, wenn die Navigation in der Seitenleiste sitzt, aber was ist, wenn dem nicht so ist? Was ist, wenn die Hauptlevel-Navigation in einer horizontalen

Toolbar oben auf der Seite sitzen soll? Gute Nachrichten: Mit ein paar kleinen Style-Anpassungen und einigen Extra-Regeln können wir genau das mit dem gleichen Markup erreichen, das wir schon benutzen.

6.5.1 Neuausrichtung

Bevor wir irgendwas anderes anfangen, wollen wir die Styles überarbeiten, damit die Links der Hauptmenüpunkte als horizontale Toolbar angezeigt werden. Dafür braucht es nur sehr wenige Änderungen. Zuerst werden die Deklarationen für die Breite aus den Regeln `div#nav` und `div#nav ul` entfernt.

```
div#nav {float: left; margin: -1px 0 0 -1px;
  background: #FDD;}
div#nav ul {margin: 0; padding: 0; background: white;
  border: 1px solid #AAA;}
```

Dies legt die Größe der `div`s und der Listen innerhalb des `div`s automatisch fest. Keine Sorge – bei den Menüs selbst werden wir das gleich reparieren. Allerdings geht das mit der horizontalen Toolbar im Moment nicht, es sei denn, wir lassen die Toolbar selbst ihre eigene Breite setzen.

Um die Toolbar oben quer über den Haupttext zu platzieren, werden wir ihren linken Rand ändern.

```
div#nav {float: left; margin: -1px 0 0 7em;
  background: #FDD;}
```

Dies wird die linke Kante der Toolbar über eine Distanz schieben, die genau der Breite des linken Randes des Haupttexts entspricht.

Weil das Navigations-`div` ein Float ist und keine explizite Breite aufweist, passt es sich selbst in der Größe derart an, dass es genau um seinen Inhalt passt. (Dies nennt man auch »shrink-wrapping«). Falls wir sie »floaten« – und das machen wir auch gleich –, gilt das Gleiche auch für die Listenelemente in den Menüs, einschließlich derer auf der obersten Ebene. Für einige Anwendungen mag das akzeptabel sein, aber hier wollen wir, dass alle die gleiche Breite haben. Warum nicht 7em? Das hat bisher auch sehr gut funktioniert.

```
div#nav li {position: relative; list-style: none; margin: 0;
  float: left; width: 7em;
  border-bottom: 1px solid #CCC;}
```

Warum haben wir die Listenelemente »gefloatet«? Damit sie sich horizontal ausrichten. Hätten wir sie nicht »gefloatet«, wäre wieder die gleiche vertikale Link-Liste herausgekommen wie vorher.

Dass alle Listenelemente die gleiche Breite haben sollen, hat folgenden Grund: Die Menüs, die aus der obersten Ebene ausklappen, müssen alle eine konsistente Breite haben. Wenn das nicht der Fall ist, könnte das Menü breiter oder schmaler sein als der Link darüber, über den es aufgerufen wurde. Also werden wir den Listen der zweiten

Ausgemerzt

Wir können hier keine Änderungen des Codes sehen, da wir einfach ein paar Sachen herausgenommen haben. Vergleichen Sie diesen Code-Abschnitt mit der gleichen Regel im Listing 6.4, wenn die Änderungen nicht nachvollziehbar sind.

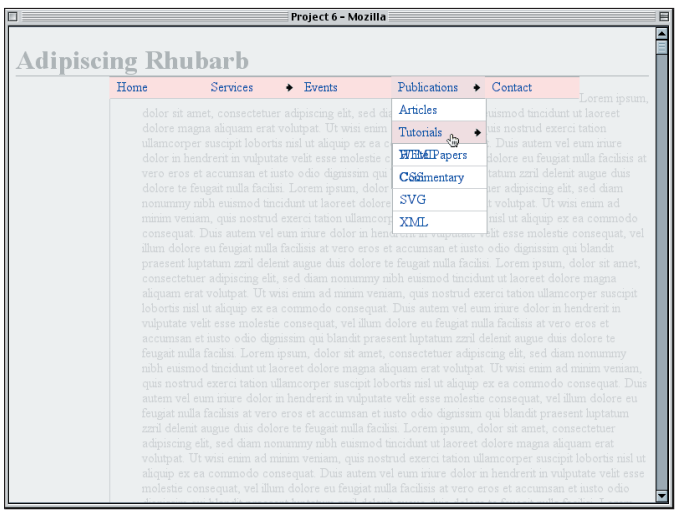
oder den weiter darunter liegenden Ebenen eine Breite über die Regel `div#nav ul ul` zuweisen und nehmen gleichzeitig die `top`- und `left`-Deklarationen heraus.

```
div#nav ul ul {position: absolute; width: 7em;
  display: none;}
```

Diese wenigen kleinen Änderungen führen zusammengenommen zum Ergebnis aus Abbildung 6.12.

ABBILDUNG 6.12

Die Menüstruktur wird als horizontale Toolbar neu angeordnet.



Bei der Abbildung 6.12 gibt es sechs Sachen anzumerken:

- Der Hauptinhalt fließt um das gefloatete `div`. Er soll auf jeden Fall soweit verschoben werden, dass er unter der Toolbar steht.
- Der hellrote Hintergrund des Navigations-`div`s ist wieder da. Das liegt daran, dass wir es mit einer unsortierten Liste zu tun haben, deren Inhalt nicht normal fließt; alle ihre Listenelemente sind gefloatet, und alle daraus hervorgehenden Listen sind positioniert. Darum ist das `ul` auf eine Höhe von Null in sich zusammengefallen, und der Hintergrund des `div`s wird sichtbar.
- Aus den gleichen Gründen besitzt das `ul`-Element der obersten Ebene keine Höhe, und die dicke graue Leiste am oberen Rand besteht eigentlich aus den oberen und unteren Rahmen des `ul`, die ganz dicht nebeneinander stehen. Das `div` erstreckt sich um die Listenelemente, weil es gefloatet ist, und Floats dehnen sich aus, um sich den in ihnen enthaltenen Floats anzupassen.
- Der Rahmen entlang der unteren Kante der Toolbar wird durch den unteren Rahmen der Listenelemente geschaffen.

- Die Menüs der zweiten Ebene erscheinen genau dort, wo wir sie haben wollen. Das liegt daran, dass sie immer noch unter Berücksichtigung der in ihnen enthaltenen Listenelemente positioniert werden. Dass sie an der unteren Kante der Listenelemente erscheinen, liegt daran, dass ihr `top` den Wert `auto` hat. (Erinnern Sie sich daran, dass wir die Deklaration `top` herausgenommen haben, und der Standardwert für `top` ist `auto`. In diesem Fall wird der obere Rand des positionierten Elements dort platziert, wo er ohne Positionierung des Elements gewesen wäre. Also tauchen die Untermenüs genau dort auf, wo sie ohne Positionierung ebenfalls aufgetaucht wären.
- Die im vorigen Punkt angesprochene automatische Platzierung passiert genauso bei den Unter-Untermenüs, also überlappen sie als Folge ihre übergeordneten Untermenüs. Das müssen wir reparieren.

Wir wollen diese Punkte der Reihe nach abarbeiten.

6.5.2 Ab in die Werkstatt

Die am leichtesten zu reparierende Sache ist, dass der Hauptinhalt direkt nach der Toolbar beginnt – ohne einen Zeilenumbruch. Anstatt `clear` zu benutzen, mit dem das gesamte Inhalts-`div` (einschließlich des linken Rahmens) heruntergeschoben werden würde, werden wir einfach das obere Padding aufstocken. Die Listenelemente sind 1,5 em groß, also nehmen wir etwas mehr em.

```
#main {color: #CCC; margin-left: 7em; padding: 2em 0 1px 5%;
  border-left: 1px solid;}
```

Damit wird die obere Kante des Inhalts des ersten Absatzes um 3em hinunter geschoben, weil das `div` zwei 2em oberes Padding hat und der Absatz einen oberen Rand von 1em.

Und nun weg mit dem roten Hintergrund. Wir wollen einen Rahmen, der sich um die Links in der Toolbar erstreckt, und alles soll einen weißen Hintergrund bekommen. Weil sich das `div` selbst bereits um die Links der obersten Ebene erstreckt, werden wir seinen Hintergrund ändern und einen Rahmen einfügen (siehe Abbildung 6.13).

```
div#nav {float: left; margin: -1px 0 0 7em;
  background: #FFF; border: 1px solid #AAA;}
```

Haben Sie den dicken, grauen Rahmen bemerkt, der immer noch oben an der Toolbar erscheint? Das ist der Rahmen von `ul`, der sich da immer noch herumtreibt. Wir könnten den Rahmen von den `ul`-Elementen insgesamt entfernen, aber das führte zu einem weiteren Problem: Die Untermenüs hätten keinen Rahmen.

Die siebte Sache

Beim IE/Win gibt es noch eine weitere Sache anzumerken: Der linke Rand der Navigationsleiste hat sich verdoppelt. Das liegt an einem Bug im IE/Win, der die Ränder bei Floats aus keinem erkennbaren Grund verdoppelt. Weil der einzige Unterschied zwischen den Abbildungen und dem IE/Win darin besteht, wie die Navigationslinks platziert sind, werden wir diesen Fehler bis zum Ende des Projekts ignorieren.

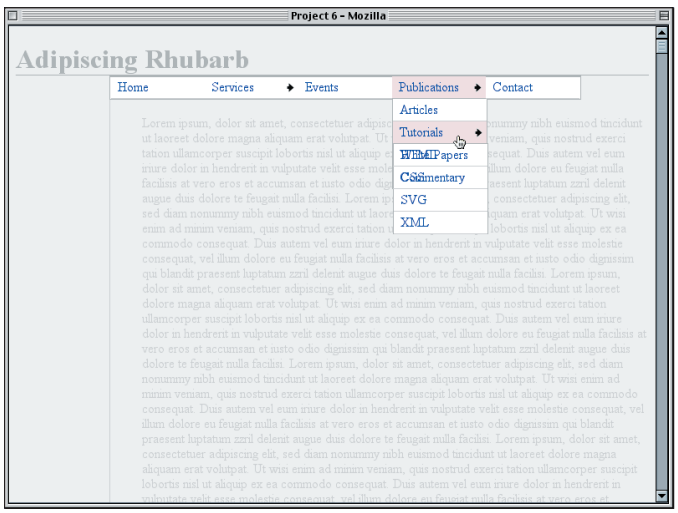


ABBILDUNG 6.13

Der Hauptinhalt wird an der Toolbar vorbeigeschoben und das Navigations-`div` erhält Hintergrund und Rahmen.

Stattdessen hellen wir den Rahmen einfach ein wenig auf und entfernen die oberen und seitlichen Rahmen, indem wir deren Breite auf Null setzen.

```
div#nav ul {margin: 0; padding: 0; background: white;
  border: 1px solid #CCC; border-width: 0 1px;}
```

So behalten die Untermenüs immer noch ihre linken und rechten Rahmen, und was das `ul` der obersten Ebene betrifft: Da wird kein Rahmen erscheinen, weil die Rahmen, die eine Breite besitzen, keine Höhe haben (weil das Element selbst keine Höhe aufweist).

Und was ist nun mit den unteren Rahmen der Untermenüs? Um die kümmert sich der untere Rahmen der Listenelemente. Optisch macht das keinen Unterschied. Wo der untere Rahmen von Listenelementen zu einem Problem werden könnte, das ist an der unteren Kante der Toolbar, wo die Rahmen der Listenelemente direkt an den unteren Rahmen des Navigations-`div`s stoßen. Die können wir loswerden, indem wir die Rahmen-Deklaration aus der Regel `div#nav li` entfernen.

```
div#nav li {position: relative; list-style: none; margin: 0;
  float: left; width: 7em;}
```

Aber das wird auch die unteren Rahmen aus den Listenelementen des Untermenüs entfernen, und wir haben uns schon darauf festgelegt, dass wir sie brauchen. Also werden wir die Rahmen für Untermenüs mit dieser Regel wieder einfügen.

```
div#nav ul ul {position: absolute; width: 7em;
  display: none;}
div#nav ul ul li {border-bottom: 1px solid #CCC;}
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
```

Auf der obersten Ebene gibt es noch eine Sache, die wir reparieren müssen: das Pfeil-Bild. Dass wir hier immer noch einen nach rechts zeigenden Pfeil haben, macht keinen Sinn mehr, weil die Menüs nach unten ausklappen und nicht nach rechts ausfahren. Da wir nun das Bild eines nach unten zeigenden Pfeils haben (das wir `dropmenu.gif` nennen können), brauchen wir nur eine einzige Regel ändern (siehe Abbildung 6.14).

```
div#nav li.submenu {background: url(dropmenu.gif) 95% 50% no-repeat;}
```

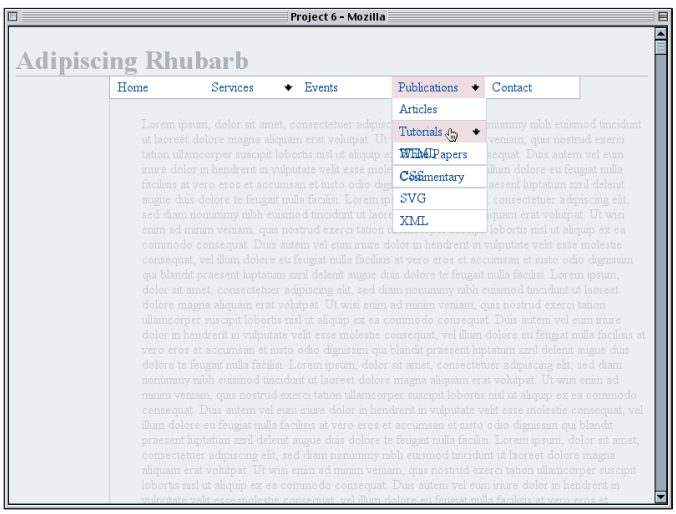


ABBILDUNG 6.14
Korrekturen an den Rahmen und neues Pfeil-Bild

6.5.3 Korrekturen am Untermenü

Wie Sie wahrscheinlich in Abbildung 6.14 bemerkt haben, ist auf der obersten Ebene alles in Ordnung, was man von den Untermenüs leider nicht behaupten kann. Zum einen weist der Eintrag *Tutorials* dank unserer letzten Änderung einen nach unten zeigenden Pfeil auf. Zum anderen überlappt das Untermenü von *Tutorials* immer noch das Ende des *Publications*-Menüs. Beides muss repariert werden.

Den Pfeil kriegen wir leicht repariert. Nun ja, *beides* ist in Wirklichkeit schnell geradezubiegen, aber das mit dem Pfeil geht einfacher. Wir brauchen nur das originale Hintergrund-Bild (`submenu.gif`) in den Hintergrund von jedem Untermenü-Listenelement zu geben, das von einem anderen Untermenü-Listenelementen abstammt. Das wirkt sich nicht auf die Links der obersten Ebene aus, die Untermenüs enthalten, weil sie nicht von anderen Untermenü-Listenelementen abstammen.

```
div#nav ul ul li {border-bottom: 1px solid #CCC;}
div#nav li.submenu li.submenu {background-image: url(submenu.gif);}
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
```

Zumindest wäre das alles, was wir machen müssten, gäbe es da nicht diesen Bug in dem Skript, das wir für den IE/Win benutzen. Offenbar wird da ein Reset auf den Standard für alle Hintergrund-Styles durchgeführt, wenn der Mauszeiger auf einem Element ruht; wenn wir also bei der letzten Änderung bleiben, dann heißt das, dass sich der Pfeil kachelförmig im ganzen Hintergrund des Untermenüs wiederholen wird. Also sagen wir ihm, was er zu tun hat, indem wir alle Werte neu deklarieren.

```
div#nav li.submenu li.submenu {background: url(submenu.gif) 95% 50% no-
repeat;}
div#nav li.submenu li.submenu:hover {background-color: #EDD;}
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
```

Somit haben wir ein Problem weniger.

Jetzt geht es an die Platzierung der Unter-Untermenüs. Zuerst einmal wollen wir die Untermenüs platzieren. Wir wollen, dass ihre obere Kante sich am unteren Rand der Top-Level-Listenelemente ausrichtet, wie sie das jetzt auch schon machen, allerdings wollen wir dieses Verhalten nun explizit einfordern. Weiterhin wollen wir, dass die linke Seite des Untermenü-Inhalts sich mit der linken Kante des Listenelement-Inhalts ausrichtet. Weil das Untermenü einen linken Rahmen hat und die Listenelemente der obersten Ebene nicht, ist der Inhalt des Untermenüs tatsächlich einen Pixel weiter rechts. Also werden wir zur Kompensierung einen negativen `left`-Wert verwenden.

```
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
div#nav ul.level2 {top: 1.5em; left: -1px;}
</style>
```

Der Wert für `top` ist so eingestellt, dass er der Summe des `font-size` der Top-Level-Listenelemente und ihrem oberen und unteren Padding entspricht, aber nur, wenn die Höhe der Zeile eines jeden Listenelements genau 1em beträgt. Das haben wir noch nicht explizit gesagt und müssen es daher nachholen. Also schreiben wir jetzt:

```
div#nav li {position: relative; list-style: none; margin: 0;
  float: left; width: 7em; line-height: 1em;}
```

Jetzt sollen die Unter-Untermenüs korrekt positioniert werden. Bei denen sollen die oberen Kanten aneinander ausgerichtet sein und das Unter-Untermenü direkt links neben seinem übergeordneten Eintrag erscheinen. Also holen wir uns das `left: 7em;` zurück, das wir in der ersten Hälfte des Projekts verwendet hatten, und nehmen einen negativen Wert für `top`, um das Menü an die gewünschte Stelle hochzuschieben.

```
div#nav ul.level2 {top: 1.5em; left: -1px;}
div#nav ul.level3 {top: -1px; left: 7em;}
</style>
```

Eine Sache noch, und wir sind fertig. Erinnern Sie sich an die Änderung der Rahmen, durch die die oberen Rahmen bei den Untermenüs entfernt wurden? Das gilt ebenso für die Unter-Untermenüs, also werden wir diese Änderung verwenden, um das Ergebnis aus Abbildung 6.15 zu bekommen.

```
div#nav ul.level3 {top: -1px; left: 7em;
  border-top: 1px solid #CCC;}
```

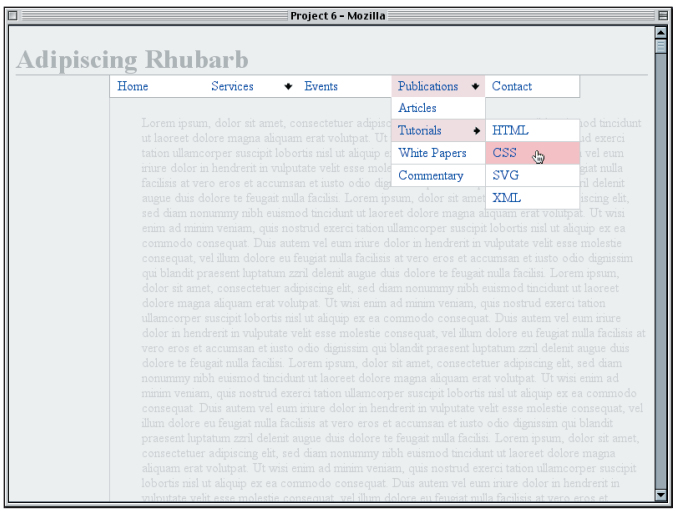


Abbildung 6.15

Untermenüs und deren Untermenüs werden aufpoliert.

Damit sind wir bei dem Stylesheet aus Listing 6.5 angekommen.

Listing 6.5 Stylesheet mit einer Toolbar in einem Menüsystem

```
body {background: #EEE; color: #000;
  behavior: url(csshover.htc);} /* WinIE behavior call */
h1 {color: #AAA; border-bottom: 1px solid; margin-bottom: 0;}
#main {color: #CCC; margin-left: 7em; padding: 2em 0 1px 5%;
  border-left: 1px solid;}
div#nav {float: left; margin: -1px 0 0 7em;
  background: #FFF; border: 1px solid #AAA;}
div#nav ul {margin: 0; padding: 0; background: white;
  border: 1px solid #CCC; border-width: 0 1px;}
div#nav li {position: relative; list-style: none; margin: 0;
  float: left; width: 7em; line-height: 1em;}
div#nav li:hover {background: #EBB;}
div#nav li.submenu {background: url(dropmenu.gif) 95% 50% no-repeat;}
div#nav li.submenu:hover {background-color: #EDD;}
div#nav li a {display: block; padding: 0.25em 0 0.25em 0.5em;
  text-decoration: none; width: 6.5em;}
div#nav>ul a {width: auto;}
div#nav ul ul {position: absolute; width: 7em;
  display: none;}
div#nav ul ul li {border-bottom: 1px solid #CCC;}
div#nav li.submenu li.submenu {background-image: url(submenu.gif);}
div#nav ul.level1 li.submenu:hover ul.level2,
div#nav ul.level2 li.submenu:hover ul.level3 {display:block;}
div#nav ul.level2 {top: 1.5em; left: -1px;}
div#nav ul.level3 {top: -1px; left: 7em;
  border-top: 1px solid #CCC;}
```

6.6 Überlegenswerte Dinge

Es gibt da einige Sachen, die Sie sich überlegen sollten, wenn Sie über CSS solche Menüs wie die aus diesem Projekt implementieren wollen. Wir lassen einige weiterreichende Fragen außen vor, z.B. ob Drop-down-Menüs eine gute Benutzeroberfläche sind (manche sagen ja, andere lehnen sie ab), und betrachten das tatsächliche Verhalten dieser Menüs.

Weil das Erscheinen von Untermenüs davon abhängt, ob Listenelemente sich in einem Rollover-Status befinden, kann es den Usern leicht passieren, dass ihnen die Menüs unabsichtlich wieder zuklappen. In dem Moment, wo der Mauszeiger sich aus einem der Menüs hinausbewegt, werden alle aktuell sichtbaren Untermenüs verschwinden. Das Gleiche gilt auch oft für mit JavaScript gebauten Menüs, aber in JavaScript können Sie eine Zeitverzögerung einbauen, so dass dem User ein Moment Zeit bleibt, um die Maus wieder in die Menüs zurückzubewegen, bevor sie verschwinden. CSS bietet keine Möglichkeit, solch eine Zeitverzögerung zu definieren, und es ist recht unwahrscheinlich, dass das je (wenn überhaupt) kommen wird.

Aus diesem Grund ist es zwingend erforderlich, dass Untermenüs wenigstens direkt an ihre übergeordneten Listenelemente angrenzen oder sie ein bisschen überlappen. Wenn eine auch nur einen Pixel große Lücke zwischen einem Menüeintrag und seinem Untermenü erscheint, ist die Wahrscheinlichkeit groß, dass der Mauszeiger den Rollover-Status verlässt, wenn er durch die Lücke wandert, und das führt zum Verschwinden aller offenen Untermenüs. So etwas macht es im Grunde unmöglich, einige (oder alle) der Untermenüs zu verwenden.

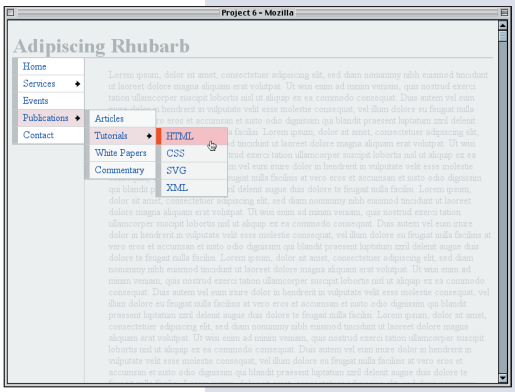
Natürlich ist es möglich, die hier präsentierten Techniken zu verwenden, um das Erscheinen von Menüs zu definieren, und dann über JavaScript deren dynamisches Verhalten zu steuern – also das Erscheinen und Verbergen von Menüs, etwa auch mit Zeitverzögerungen zur einfacheren Nutzung der Menüs, und das Event-Handling, was alles zusammenfügt. Eine solche Technik sprengt den Rahmen dieses Buches, aber im Wesentlichen gehört dazu, die Regel `display: block;` und die `behavior`-Deklaration wegzulassen und sich darauf zu verlassen, dass JavaScript den Platz dieser Styles einnimmt.

Und was nun jeweils besser ist, darüber gehen die Meinungen auseinander, und wir werden uns hier zu keiner Entscheidung erdreisten. Einige sagen, dass Verhalten nicht über CSS gesteuert werden sollte, da es eine Präsentationssprache ist, sondern stattdessen über eine Scripting-Sprache wie JavaScript. Andere sind der Meinung, dass diese Art von Verhalten mit CSS realisierbar ist, also sei an diesen Techniken nichts Verkehrtes. Diese Debatte wird sich wohl noch lange hinziehen, also müssen die Autoren entscheiden, was sie bevorzugen, und sich entsprechend verhalten.

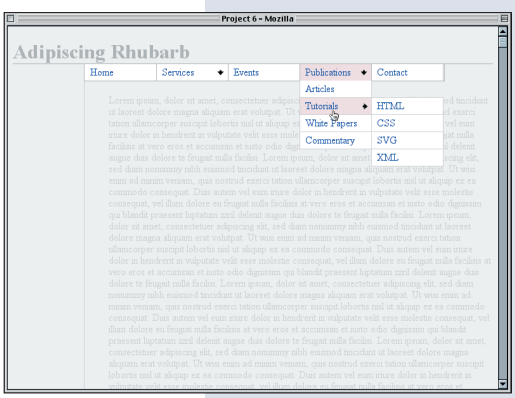
6.7 Spielwiese

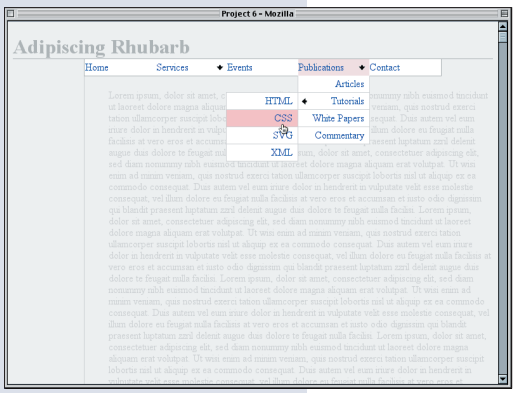
Jetzt, wo Sie wissen, wie Drop-down-Menüs mit CSS erstellt werden, sollten Sie versuchen, sie auf die folgenden Arten auf Vordermann zu bringen:

1. Nehmen Sie das Stylesheet aus der ersten Projekthälfte und ändern Sie es derart ab, dass die Einträge einen grauen Balken auf der linken Seite bekommen, aber versuchen Sie, das ohne ein Hintergrundbild hinzukriegen. Der Balken soll bei dem Menüeintrag, über dem der Mauszeiger gerade ruht, auf rot wechseln. Da Sie gerade dabei sind, machen Sie bei jedem Untermenü-Level den Hintergrund etwas dunkler.



2. Erweitern Sie die Styles aus der zweiten Projekthälfte, so dass die grauen Rahmen die Links der obersten Ebene trennen. Die Herausforderung besteht darin, dass Sie das ohne irgendwelche Änderungen des optischen Erscheinens der Untermenüs schaffen.





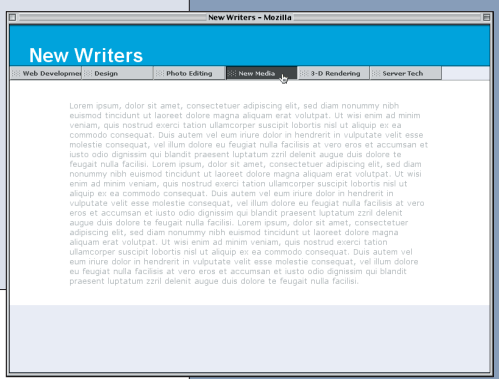
3. Überarbeiten Sie die Styles aus der zweiten Hälfte des Projekts, damit die Unter-Untermenüs sich nicht mehr nach rechts aus dem sie aufrufenden Menü öffnen, sondern nach links. Damit das so gut wie möglich aussieht, richten Sie in den Untermenüs den Text rechtsbündig aus, aber achten Sie darauf, dass er nicht gegen die rechte Kante der Untermenüs stößt. Und Sie brauchen auch noch eine neue Pfeil-Grafik. (Spiegeln Sie `submenu.gif` in einem Bildbearbeitungsprogramm wie Photoshop einfach horizontal.)

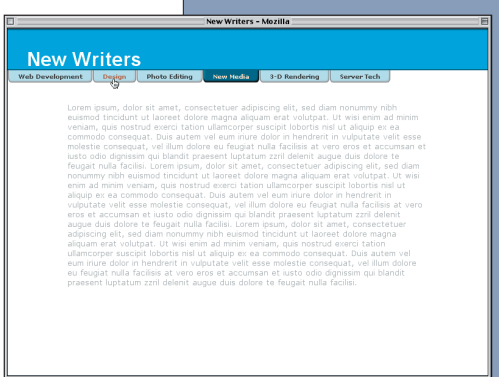
7

Hereinspaziert! Attraktive Tabs auf Ihrer Website

Willkommen allezeit bei Göttern und den Menschen ist jener, der sich selbst hilft. Für ihn werden alle Türen aufgestoßen: ihn grüßen alle Zungen, ihn krönen alle Ehren, alle Augen folgen ihm mit Verlangen.

Ralph Waldo Emerson





Wir beobachten zunehmend einen Trend in Richtung Listen, die »Menüs« (also Sammlungen von Links) repräsentieren, und das große Interesse daran, nicht nur Links in eine Seitenleiste zu stellen, sondern auch horizontale Reihen von Links einzurichten. Ein gutes Beispiel solcher Links finden Sie oben auf der Website von Amazon.com oder bei Apple.com. Solche Sammlungen werden optisch oft Karteireitern nachgebildet, sogenannten Tabs, obwohl das nicht immer der Fall ist.

Für längere Zeit bedeutete das Umwandeln einer unsortierten Liste in eine Reihe von Tabs, dass ein – tja, »Kästchen-Design« dabei herauskam. Und ein wenig langweilig war es obendrein. Das lag daran, dass die Tabs normalerweise durch Einrahmen der Listenelemente oder der Links selbst erstellt wurden. Für ein erstes Design ist das okay, aber wenn Sie einen professionellen Look wollen, lässt das Ergebnis zu wünschen übrig.

Dann bahnte Douglas Bowman (`http://www.stopdesign.com/`), wohl am bekanntesten durch seine tabellenfreie Neugestaltung von *Wired News*, den Weg für einen neuen Ansatz, über den Tabs erstellt werden konnten, deren optisch verblüffende Wirkung nur durch die Fantasie der Autoren begrenzt wurde. Er beschrieb diese Technik zuerst in dem Artikel »Sliding Doors of CSS« (`http://alistapart.com/articles/slidingdoors/`) und führte diese Technik dann detaillierter in einem weiteren Artikel aus. In diesem Projekt werden wir einige Varianten von Dougs ursprünglicher Idee ausprobieren, aber der Kern der Idee ist ganz sein eigen.

7.1 Projektziele

Dank der hervorragenden Leistungen unseres Vertriebsteams haben wir einen Vertrag über die Gestaltung einer Website für einen neuen Verlag namens »New Writers« abschließen können. Als Ziele für diese Phase des Projekts legen wir Folgendes fest:

- Wir brauchen eine Vorlage, die zeigt, wie die unveränderlichen Navigationselemente aussehen werden. Diese Vorlage soll ein paar grundlegende Styles einsetzen, die bereits ausgearbeitet wurden, insbesondere für den Seitentitel.
- Die Navigation wird als unsortierte Liste im HTML geschrieben sein, aber sie soll als eine horizontale Reihe von Buttons oder Tabs präsentiert werden, die zwischen dem Seitentitel und dem Hauptinhalt steht.
- Die Buttons/Tabs sollen visuell ansprechend sein. Der Kunde hat schon eine Menge mit CSS erstellte Buttons gesehen, die letzten Endes einfach nur Rechtecke mit Hintergrundfarbe waren, aber für seine Site wünscht er etwas Besseres.
- Egal für welches Aussehen wir uns bei den Buttons/Tabs entscheiden, sie müssen leicht änderbar oder aktualisierbar sein, falls das Design der Seite sich ändert.

Also werden wir uns darauf konzentrieren, die Navigationselemente in etwas Ansprechendes zu verwandeln. Das ist natürlich nicht der vollständige Kundenauftrag; beim Design einer vollständigen Site fällt noch eine Menge mehr Arbeit an. Aber für die Zwecke dieses Projekts ist es dennoch ausreichend, dass wir uns auf die Erstellung einer grundlegenden Vorlage für das Styling der Navigationselemente konzentrieren.

7.2 Vorbereitungen

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

Laden Sie die Dateien für Projekt 7 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch07proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und erneut laden.

7.3 Die Grundlagen

An der Vorlagenseite für unser Projekt ist nicht viel dran, wie in Abbildung 7.1 zu sehen ist.

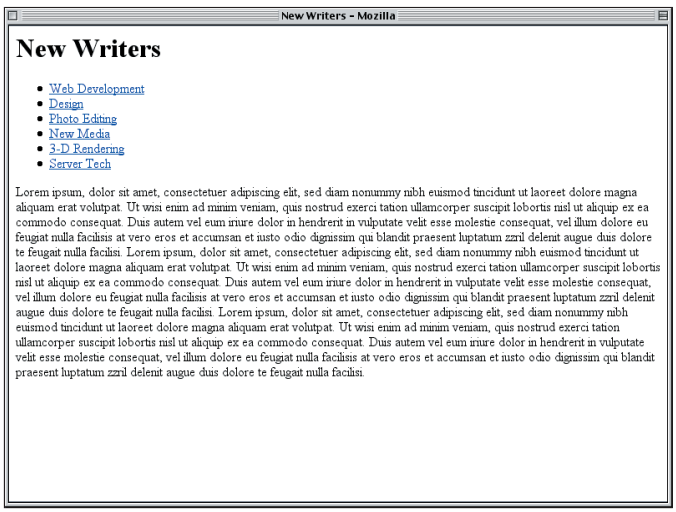


ABBILDUNG 7.1

Das ungestylte Vorlagendokument

Bevor wir uns an die Arbeit mit den Navigationselementen machen, wollen wir den Rest der Vorlage stylen. Das wird uns einen Rahmen für die Styles unserer Buttons geben.

Der erste Schritt wird darin bestehen, das gesamte Dokument selbst zu stylen, d.h. »globale« Styles zu schreiben, die für das Dokument und alle seine Bestandteile gelten. Das machen wir, indem wir den gesamten »Fugen«- oder »Gutter«-Raum an den Rändern des Dokuments entfernen, Vorder- und Hintergrundfarben einstellen und eine Liste möglicher Fonts für die `html`- und `body`-Elemente definieren.

```
<style type="text/css">
html, body {margin: 0; padding: 0;
  color: #000; background: #EEF;
  font-family: Verdana, Arial, sans-serif;}
</style>
```

Warum haben wir beide Elemente gestylt? Aus dem gleichen Grund, warum wir das Padding und die Ränder auf Null gesetzt haben – wir wollen auf Nummer sicher gehen. Obwohl alle bekannten Browser Abstände in einem Dokument, also das »Gutter Spacing«, entweder mit Rändern oder Padding beim `body` erzwingen, ist es möglich, dass ein Browser stattdessen das Padding auf das `html`-Element anwendet. Da wir, um sicherzugehen, nur sechs Zeichen mehr brauchen, geben wir unserem Cocktail noch `html` hinzu. Wie in der Abbildung 7.2 zu sehen ist, führt es genau zu dem gewünschten Ergebnis.

Fugen wegputzen

Die meisten Browser erzwingen den »Gutter« (die »Fugen« um den Text herum) über Ränder beim `body`, aber Opera besorgt das über Padding beim `body`. Das Verhalten von Opera ist wahrscheinlich korrekter, aber diese Inkonsistenz hat uns dazu gebracht, zum Entfernen der »Fugen« sowohl `padding` als auch `margin` zu setzen.

ABBILDUNG 7.2

Das Dokument bekommt »globale« Styles

Jetzt wollen wir unsere Aufmerksamkeit dem Hauptinhalt zuwenden. Wir finden hier im Wesentlichen Blindtext vor, der Platz schaffen soll, ohne wirklich maßgeblich für das tatsächliche Aussehen der Seite zu sein. Nichtsdestotrotz werden wir die Schriftgröße auf klein setzen, weil das auch für das Endprodukt gelten soll. Wir werden auch einen weißen Hintergrund und einen hellgrauen Vordergrund wählen. Dies wird die visuelle Präsenz des Textes abschwächen, ohne dass er völlig aus dem Blick gerät.

```
html, body {margin: 0; padding: 0;
  color: #000; background: #EEF;
  font-family: Verdana, Arial, sans-serif;}
#main {font-size: small; color: #AAA; background: #FFF;}
</style>
```

Das einzige echte Problem besteht darin, dass der Text sich bis zu den Kanten des weißen Hintergrunds erstreckt. Weil er nur als Platzhalter dient, wollen wir zwischen dem Text und den Kanten des Haupt-`div`s etwas Platz einfügen. Das können wir über etwas Padding für das `div`-Element selbst erledigen – siehe Abbildung 7.3.

```
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%;}
```

Nun wenden wir uns dem Titel »New Writers« zu. Die Marketing-Abteilung hat uns darauf hingewiesen, dass wir den weißen Text vor einem ganz speziellen Blau platzieren sollen, also stellen wir das gleichzeitig mit der Schriftgröße und -familie ein.

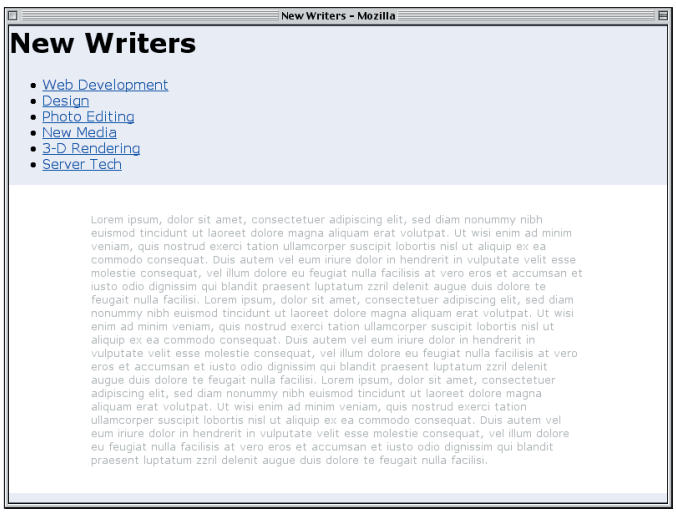


Abbildung 7.3

Der Hauptinhalt nimmt langsam Form an.

```
html, body {margin: 0; padding: 0;
  color: #000; background: #EEF;
  font-family: Verdana, Arial, sans-serif;}
h1 {color: #FFF; background: rgb(0%,56%,84%);
  font: bold 200%/1em Arial, Verdana, sans-serif;}
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%;}
```

Obendrein wollen wir nicht, dass der Text so eng an der Kante des blauen Hintergrunds sitzt, und die Überschrift soll oben an der Kante der Seite stehen. Das können wir beides über Einfügen von Padding und Entfernen der Ränder regeln. Wo wir schon gerade dabei sind, wollen wir oben und unten beim `h1` einen Rahmen mit zwei Pixeln einsetzen. Darüber wird eine gewisse optische Trennung erzielt – siehe Abbildung 7.4.

Abbildung 7.4

Der Titel wird von der Seite abgesetzt.

Höhe im Font

Sie haben wohl schon den Teil `200%/1em` der Deklaration bemerkt. Damit wird die Schriftgröße auf `200%` und die `line-height` auf `1em` eingestellt. Beachten Sie, dass wenn Sie die `line-height` über eine `font`-Deklaration einstellen wollen, sofort danach der Wert für `font-size` folgen und mit einem Slash (/) getrennt werden *muss*.

```
h1 {color: #FFF; background: rgb(0%,56%,84%);
  font: bold 200%/1em Arial, Verdana, sans-serif;
  padding: 1em 1em 0; margin: 0;
  border: 1px solid rgb(0%,31%,46%);
  border-width: 2px 0;}
```

7.4 Styling der Links

Nachdem wir nun den Rest der Seite eingerichtet haben, können wir uns an die Navigation machen. Wir können in Abbildung 7.4 sehen, wie sie als unsortierte Liste zwischen dem Titel und dem Hauptinhalt eingezwängt sitzt. Klare Sache – das sieht richtig hässlich aus, und das werden wir nun ausbügeln!

7.4.1 Start mit den Styles

Es gibt zwei Wege, um eine Liste in eine horizontale Reihe von Links zu verwandeln. Entweder werden die Listenelemente als inline deklariert oder »gefloatet«. Beide Herangehensweisen haben ihre Vor- und Nachteile, aber einer der Hauptvorteile bei dem Weg über die Floats besteht darin, dass wir damit grafisch sehr schöne Buttons gestalten können, die in modernen Browsern, aber auch im Internet Explorer funktionieren.

Wir wollen uns nun das Markup für die Navigationselemente anschauen (siehe Listing 7.1).

Listing 7.1 Das Markup der Navigation

```
<ul id="nav">
<li><a href="/webdev/">Web Development</a></li>
<li><a href="/design/">Design</a></li>
<li><a href="/photos/">Photo Editing</a></li>
<li id="current"><a href="/newmed/">New Media</a></li>
<li><a href="/render/">3-D Rendering</a></li>
<li><a href="/server/">Server Tech</a></li>
</ul>
```

Das war's schon – mehr brauchen wir nicht. Wir werden uns später in diesem Projekt den `id`-Wert `current` zunutze machen, aber momentan brauchen wir ihn nicht.

Damit diese Technik funktioniert, werden wir zuerst dafür sorgen, dass die unsortierte Liste weder Padding noch Ränder hat. Damit wird sie sich weitgehend ähnlich wie ein `div`-Element verhalten.

```
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%;}
#nav {margin: 0; padding: 0;}
</style>
```

Jetzt können wir uns bei den Listenelementen selbst an die Arbeit machen. Wir werden sie gemeinsam als Floats nach links setzen, damit sie alle horizontal ausgerichtet sind und der erste Link am weitesten links sitzt. Wir werden – dem Explorer zuliebe – auch noch explizit die Gliederungspunkte (Bullets) entfernen.

```
#nav {margin: 0; padding: 0;}
#nav li {list-style: none; float: left; margin: 0;}
</style>
```

Wir haben hier 6 Links in der Navigation, also könnten wir sie alle gleich groß machen, indem wir 100 durch 6 teilen. Das ergibt 16 2/3, und das heißt, alle könnten eine `width` von `16.6667%` haben. Das macht die Angelegenheit aber recht wackelig – ein einziger Rundungsfehler könnte das ganze Design über den Haufen werfen, und es bleibt kein Platz, um den Listenelementen Ränder oder Rahmen hinzuzufügen (und das wollen wir gleich machen). Deshalb gehen wir auf Nummer sicher und machen die Listenelemente `15%` breit. Wir fügen einen gepunkteten roten Rahmen ein, damit besser zu sehen ist, wie sie angelegt sind – siehe Abbildung 7.5.

```
#nav li {list-style: none; float: left; width: 15%;
  border: 1px dotted red;}
```

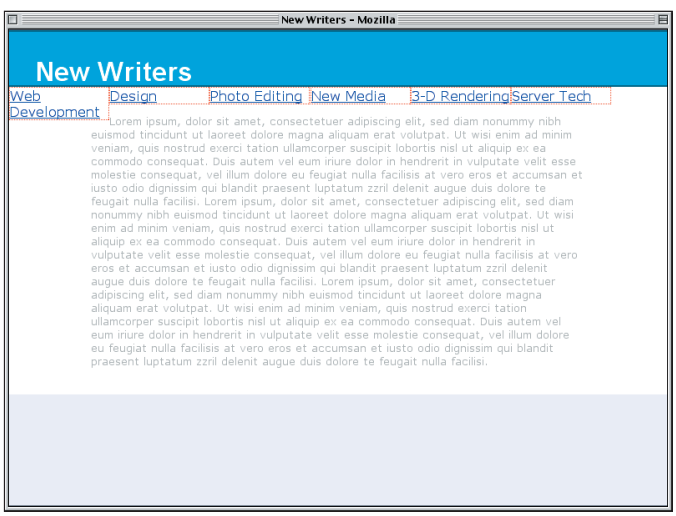


ABBILDUNG 7.5

Ein roher Anfang für die Navigations-Tabs

Die Tabs sind nun mehr oder weniger so ausgerichtet, wie wir sie haben wollen, aber es gibt da links ein Problem. Der Tab für »Web Development« wird nicht nur auf zwei Zeilen umgebrochen, sondern sorgt auch noch dafür, dass ein Teil des Haupttexts drum herum fließt.

Das kommt natürlich daher, dass wir alle Listenelemente gefloatet haben. Normal fließender Text (wie der im `div`-Element mit dem Hauptinhalt) umfließt gefloatete Elemente (wie die Listenelemente). Was vielleicht nicht so offensichtlich ist: Die unsortierte Liste, in der diese Listenelemente enthalten sind, hat eine Höhe von Null.

Punkte als Stolpersteine

Theoretisch sollte ein Listenelement als Float keine Markierung (Gliederungspunkte) generieren, aber das scheint der Explorer nicht zu wissen. Darum sollten Sie jedes Mal, wenn Sie diese Technik verwenden, `list-style` explizit auf `none` setzen.

Textumbruch

Wenn Sie in einem Browser-Fenster arbeiten, das deutlich breiter als 800 Pixel ist, werden Sie in Ihrer Projektdatei wahrscheinlich keinen Textumbruch sehen. Versuchen Sie, wenigstens vorübergehend mal das Fenster zu verkleinern.

Wenn wir ihr eine Hintergrundfarbe geben, dann würde sie nicht auf dem Bildschirm erscheinen. Also ist die obere Kante des `div`-Elements mit dem Hauptinhalt tatsächlich oben an den Navigations-Links ausgerichtet, und der weiße Hintergrund, der hinter den Links sichtbar ist, ist in Wirklichkeit der Hintergrund des Inhalts-`div`s.

Das mag vielleicht etwas merkwürdig erscheinen, aber es ist definitiv das, was passieren sollte. Weil wir das allerdings nicht haben wollen, werden wir das Inhalts-`div` unter die gefloateten Listenelemente schieben. Dies lässt sich nun auch wieder recht leicht bewerkstelligen: Wir werden einfach das `div`-Element unter den Floats bereinigen.

```
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%; clear: left;}
```

Jetzt wird die obere Rahmenkante des `div`-Elements genau unter die Randunterkante aller linksgerichteten Floats platziert, die in dem Dokument vor dem `div`-Element kommen – beispielsweise die Listenelemente. Das bedeutet, dass der Body-Hintergrund zwischen dem `h1` und dem Inhalts-`div` sichtbar sein wird, wie wir in Abbildung 7.6 sehen können, aber das macht nichts. Wir werden uns das sogar später zunutze machen.

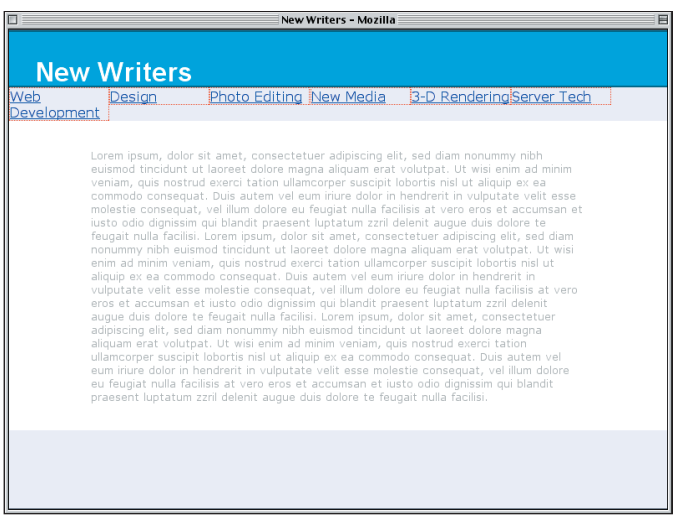


Abbildung 7.6

Der Inhalt ist unter die gefloateten Listenelemente geschoben worden, der Textfluss beginnt damit auf jeden Fall in einer neuen Zeile.

7.4.2 Überarbeitung der Tabs

Die vorangegangene Änderung hat das Problem mit dem Haupttext gelöst, der um die Links herumgeflossen ist, aber an dem Umbruch des Texts »Web Development« auf zwei Zeilen hat das nichts geändert. Das lässt sich aber leicht nachholen.

```
#nav li {list-style: none; float: left; width: 15%;
  white-space: nowrap;
  border: 1px dotted red;}
```

Dies ist dem alten HTML-Attribut `nowrap` für Tabellenzellen recht ähnlich, außer dass man mit CSS den Umbruch des Inhalts von beliebigen Elementen verhindern kann.

Bevor wir hier weitermachen, sollten wir uns überlegen, wie dieser Text verwendet werden wird. Es ist unsere Absicht, ihn über Bilder zu platzieren, die die Tabs schön aussehen lassen. Somit ist es wichtig, dass der Text eine Größe aufweist, die dem Kontext angemessen ist, in dem er erscheint (die Tabs). Weil Bildgrößen in Pixel angegeben werden, werden wir das Gleiche für den Text machen. Tatsächlich können sowohl die Schriftgröße als auch die Zeilenhöhe in Pixel eingestellt werden.

```
#nav li {list-style: none; float: left; margin: 0; width: 15%;
  font-size: 10px; line-height: 20px; white-space: nowrap;
  border: 1px dotted red;}
```

Der nächste Schritt, den wir uns überlegen müssen, bezieht sich auf die Links selbst, die in den gefloateten Listenelementen sitzen. Wir werden die Unterstriche herausnehmen und einen soliden Rahmen hinzufügen, damit wir sehen können, wie die Links in den Listenelementen angeordnet sind (siehe Abbildung 7.7).

```
#nav li {list-style: none; float: left; margin: 0; width: 15%;
  font-size: 10px; line-height: 20px; white-space: nowrap;
  border: 1px dotted red;}
#nav a {text-decoration: none; border: 1px solid;}
</style>
```

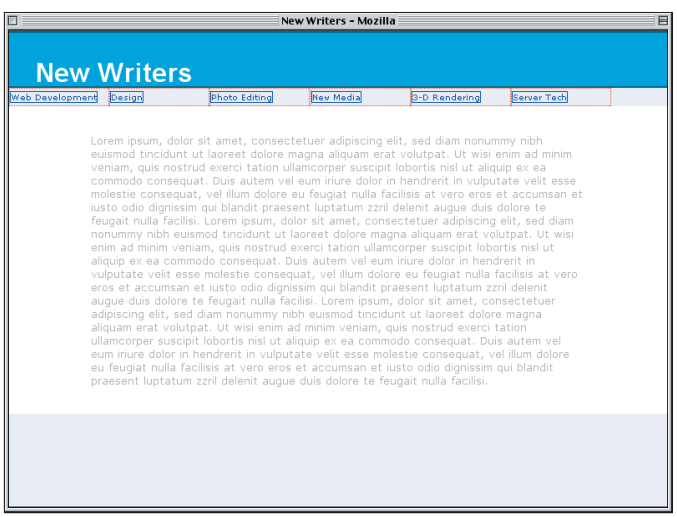


Abbildung 7.7

Die Links werden in Relation zu den Listenelementen betrachtet.

Wie wir in Abbildung 7.7 sehen können, bilden die Links kleinere Kästen innerhalb der gefloateten Listenelemente. Es wäre besser, wenn wir diese Links dazu bringen könnten, die Listenelemente vollständig auszufüllen oder wenigstens so viel davon wie möglich.

Änderung der Textgröße

Denken Sie daran, dass User bei den meisten Browsern die Textgröße verändern können – sogar Text, dessen Größe in Pixel angegeben ist. Der Internet Explorer für Windows ist die einzige Ausnahme, obgleich eine sehr weit verbreitete.

Wir wollen nun einen Moment unsere Situation durchdenken. Jedes Listenelement ist gefloatet und generiert darum eine Blocklevel-Box, sehr ähnlich wie die von einem `div`-Element geschaffene Blocklevel-Box. Tatsächlich können Elemente in diesem Listenelement nur erkennen, dass sie sich innerhalb einer Blockbox befinden. Nur Elemente außerhalb des Floats reagieren so darauf, dass sie um das Element herum fließen. Der Link ist nur ein Link, der eine Inline-Box generiert. Damit also ein Inline-Link ein Blocklevel-Listenelement ausfüllt, brauchen wir einfach nur die Links dazu zu bringen, dass sie Blockboxen generieren.

```
#nav a {display: block;
  text-decoration: none; border: 1px solid;}
```

Beachten Sie, dass wir nicht die Art der Links selbst geändert haben. Die `a`-Elemente sind immer noch Inline-Elemente. Hier sorgt CSS nur dafür, dass sie Blockboxen generieren. Dies ist ein subtiler, aber entscheidender Unterschied. Wenn CSS die Elemente selbst zu Blocklevel geändert hätte, könnte die Dokumentvalidierung leicht fehlschlagen. Das passiert allerdings nicht, da CSS sich auf die Präsentation auswirkt und nicht auf die Änderung der Dokumentstruktur.

Apropos Präsentation – die fraglichen Links sehen mit Text in normalem Gewicht etwas blass aus. Wir wollen die Links auf fettgedruckt einstellen, damit sie ein wenig besser hervorgehoben sind.

```
#nav a {display: block;
  text-decoration: none; font-weight: bold;
  border: 1px solid;}
```

Wir können das Ergebnis dieser beiden Änderungen in Abbildung 7.8 sehen. Beachten Sie, wie die Rahmen der Links innen genau mit den gepunkteten roten Rahmen abschließen, die wir an den Listenelementen eingerichtet haben.

So wird es jedenfalls in den meisten Browsern erscheinen. Die große Ausnahme ist der IE5/Mac, der die Links fälschlicherweise so breit macht wie die unsortierte Liste, anstatt so breit wie die Listenelemente. Das wollen wir definitiv nicht, also muss eine Lösung her, und es stellt sich heraus, dass diese darin besteht, die Links innerhalb der gefloateten Listenelemente selbst zu floaten.

```
#nav a {display: block; float: left;
  text-decoration: none; font-weight: bold;
  border: 1px solid;
```

Das biegt das Layoutproblem für den IE5/Mac wieder gerade, aber wir wollen absolut nicht, dass die Links in anderen Browsern gefloatet sind. Wir brauchen also einen Weg, wie wir dieses Floating für alle Browser außer dem IE5/Mac abschalten können. Glücklicherweise hat uns Doug Bowman diesen Weg gezeigt, als er die »Sliding Doors«-Technik erfand. Es gibt einen Parsing-Bug, der nur den IE5/Mac betrifft, und er kann dazu eingesetzt werden, die Nicht-Floaten-Regel vor dem IE5/Mac zu verstecken.

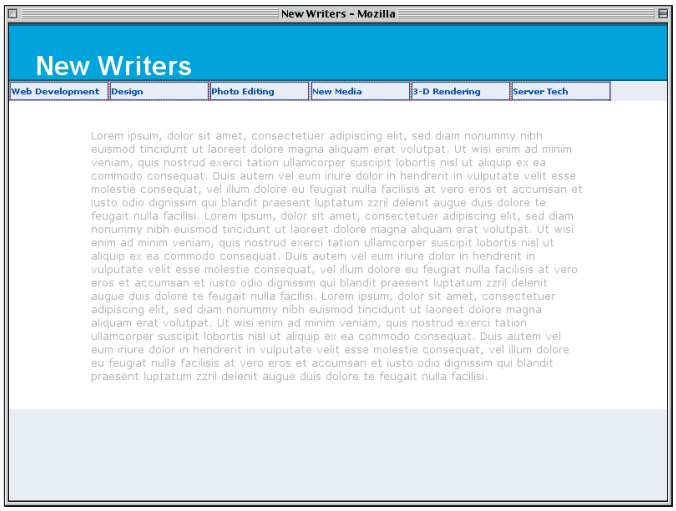


ABBILDUNG 7.8
Links in Fettdruck, die Blockboxen generieren

```
#nav a {display: block; float: left;
  text-decoration: none; font-weight: bold;
  border: 1px solid;
/* Commented Backslash Hack hides rule from IE5-Mac \*/
#nav a {float: none;}            /* End IE5-Mac hack */
</style>
```

Somit floaten alle Browser die Links (wegen `float: left`), und dann machen alle Browser außer dem IE5/Mac den Float für die Links wieder ungeschehen (dank `float: none`). Dies stellt beim IE5/Mac die Konsistenz im Layout wieder her, und wir können fortfahren.

Bevor wir damit weitermachen, die Möglichkeiten der »Sliding Doors«-Technik voll auszuschöpfen, wollen wir uns einen einfacheren Weg anschauen, wie die Links gestylt werden können, indem sie in graue, »vorspringende« Buttons verwandelt werden. Der erste Schritt besteht darin, den Vordergrund (Text) und den Hintergrund der Links in verschiedenen Grauschattierungen einzufärben.

```
#nav a {display: block; float: left;
  text-decoration: none; font-weight: bold;
  border: 1px solid;
  background: #CCC; color: #333;}
```

Jetzt brauchen wir nur noch den vorspringenden Effekt zu schaffen. Es ist bereits ein solider Rahmen vorhanden, und wir können das `solid` auf `outset` ändern. Das Problem ist, dass wir keine Kontrolle darüber haben, wie die Rahmenfarben modifiziert werden, um den `outset`-Effekt zu schaffen. Der eine Browser macht vielleicht den Rahmen oben und links weiß und den unteren und rechten Rahmen schwarz, während ein anderer feinere Schattierung wählt. Tatsächlich gibt es kaum eine browserübergreifende Konsistenz bei der Outset-Schattierung (genau wie bei den Schattierungen von `inset`, `ridge` und `groove`). Daher belassen wir den Rahmen

Wie es funktioniert

Der Schlüssel zu diesem Hack ist der Backslash (\), der direkt vor dem Sternchen am Ende des ersten Kommentars kommt. Dies lässt den IE/Mac denken, der Kommentar sei noch nicht beendet, und er nimmt an, dass die nächste Regel Teil eines Kommentars ist.

auf `solid` und deklarieren unsere eigene Schattierung, indem wir die Farbe für jede Rahmenseite einstellen.

```
#nav a {display: block; float: left;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #CCC; color: #333;}
```

Es wird Zeit, dass wir den gepunkteten roten Rahmen entfernen, also besorgen wir das auch gleich, und das Ergebnis sehen Sie in Abbildung 7.9.

```
#nav li {list-style: none; float: left; margin: 0; width: 15%;
  font-size: 10px; line-height: 20px; white-space: nowrap;}
```

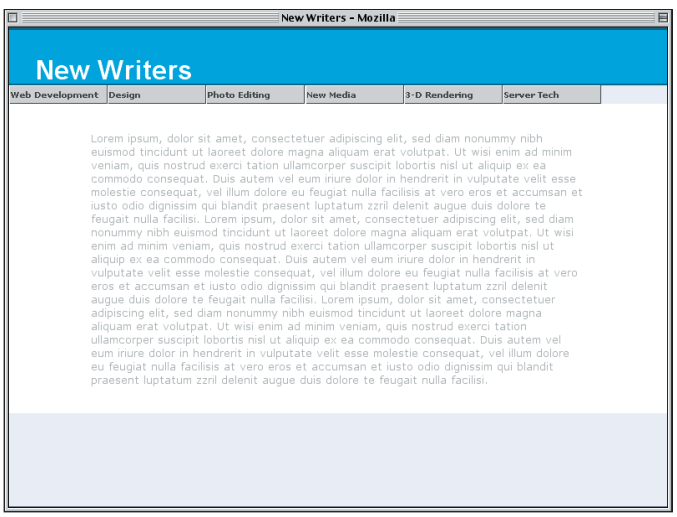


ABBILDUNG 7.9

Die Links wirken immer mehr wie Buttons.

Falls wir einmal den Kontrast zwischen den helleren und dunkleren Kanten verringern oder eine feinstufigere Färbung vornehmen wollen, brauchen wir nur die Werte für `border-color` zu ändern.

7.4.3 Einfügen von Texturen

Wir haben bisher schon ganz schön viel geschafft, aber einiges an Arbeit liegt noch vor uns. Unser größtes visuelles Problem ist momentan, dass der Text der Links direkt am linken Rahmen klebt, was nicht sehr attraktiv wirkt. Wir könnten den Links auf der linken Seite etwas Padding geben, um den Text beiseite zu schieben, und das werden wir auch bald machen.

Da wir den Text ja sowieso verschieben werden, können wir das gleich so wählen, dass genug Platz ist, um ein Bild in alle Links zu setzen. Auf diese Art können wir den Links eine visuelle Struktur geben. Schauen Sie sich beispielsweise das Bild in Abbildung 7.10 an (mit einer Vergrößerung von 1600 %).

Die großen weißen und schwarzen Pixel sind zu sehen, und der Rest des Bildes ist transparent (was durch das grau-weiße Schachbrettmuster angedeutet wird). Wenn wir dieses Bild in die Links einfügen, wird die graue Hintergrundfarbe durch die transparenten Bereiche des Hintergrundbildes sichtbar sein.

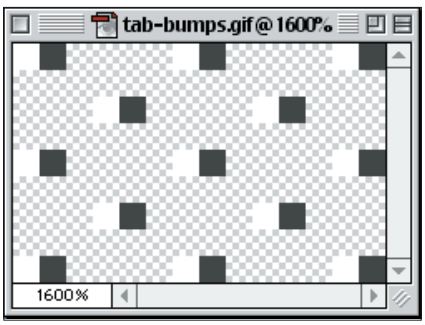


Abbildung 7.10

Mit diesem Bildchen erhält der Link-Hintergrund eine Textur.

Wir wollen, dass dieses Bild nur einmal erscheint, ein wenig von der linken Kante des Links aus eingerückt und vertikal zentriert, also schreiben wir Folgendes:

```
#nav a {display: block; float: left;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #CCC url(tab-bumps.gif) 2px 50% no-repeat;
  color: #333;}
```

Wenn wir alles so ließen, wie es jetzt ist, würde das Bild unschön hinter dem Text der Links erscheinen. Das bügeln wir aus, indem wir dem Link etwas Padding geben; im Padding wird das Bild erscheinen, aber nicht der Text. Weil das Bild 14 Pixel breit und ein wenig eingerückt ist, werden wir den linken Seiten der Links 20 Pixel Padding zufügen – das Ergebnis sehen Sie in Abbildung 7.11.

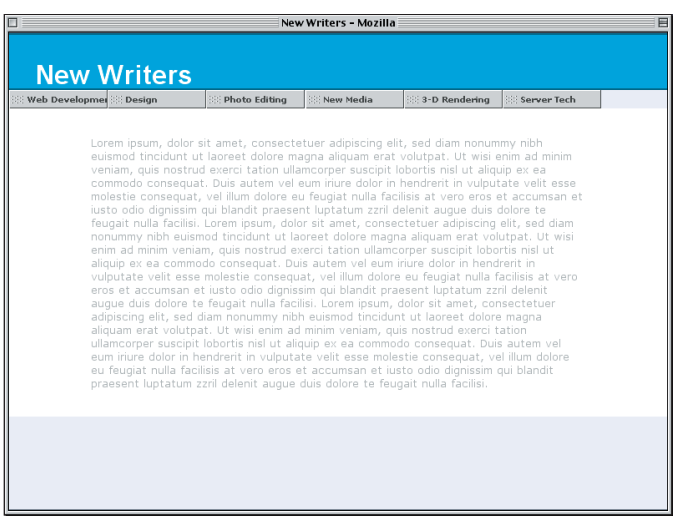


Abbildung 7.11

Über ein Hintergrundbild erhalten die Links etwas Textur.

Umgang mit Text

Wenn Sie in einem Browser-Fenster arbeiten, das deutlich breiter als 800 Pixel ist, werden Sie in Ihrer Projektdatei wahrscheinlich keinen Textumbruch sehen. Versuchen Sie, wenigstens vorübergehend das Fenster zu verkleinern. Der IE/Win wird den Text nicht beschneiden, sondern die Listenelemente so weit strecken, dass sie um den Text passen. Das klingt wie eine gute Idee, bis Sie erkennen, dass so die Wahrscheinlichkeit steigt, dass die Buttons in mehrere Zeilen umgebrochen werden.

```
#nav a {display: block; float: left; padding: 0 0 0 20px;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #CCC url(tab-bumps.gif) 2px 50% no-repeat;
  color: #333;}
```

Das sieht alles großartig aus – bis auf eine Sache: Der Text im Link »Web Development« ist so weit verschoben worden, dass er abgeschnitten wird. Für den Moment werden wir das ignorieren, aber wir kümmern uns später in diesem Projekt um einen Weg, das zu beheben.

7.4.4 Rollover-Effekte und Abschlussarbeiten

Jetzt, wo wir diese schönen Buttons haben, sollen die Links einen Rollover-Effekt bekommen. Der wohl einfachste Rollover-Effekt ist, die Farben eines Links zu »invertieren«, und das werden wir hier auch machen. Wir werden die Farben des Vordergrunds mit der des Hintergrunds vertauschen und gleichzeitig die Rahmenfarben so ändern, dass sie die invertierte Farbe ihres Nicht-Rollover-Zustands annehmen.

```
#nav a {float: none;}              /* End IE5-Mac hack */
#nav a:hover {background-color: #333; color: #CCC;
  border-color: #000 #CCC #CCC #000;}
</style>
```

In dieser Phase bekommt unser Projekt den letzten Schliff dadurch, dass wir dem `div`-Element des Hauptinhalts oben einen grauen Rahmen geben. Dadurch werden die Links sozusagen eingefasst, und das setzt sie visuell in einen Balken, in dem nur sie alleine enthalten sind (siehe Abbildung 7.12).

```
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%; clear: left;
  border-top: 1px solid gray;}
```

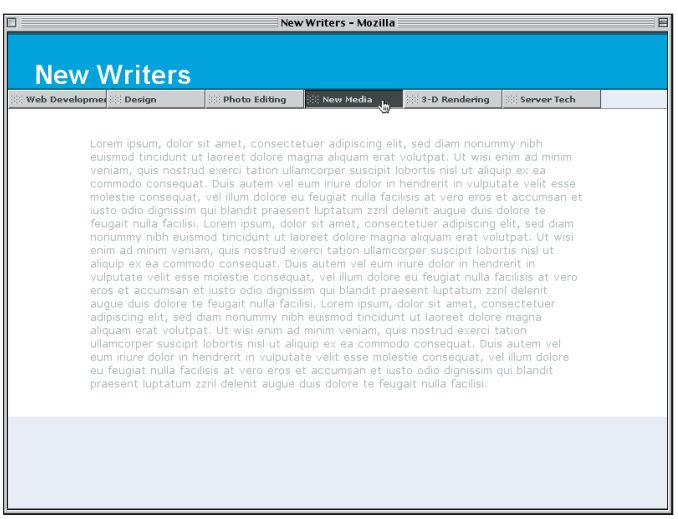


ABBILDUNG 7.12

Rollover-Effekte und ein Rahmen für den Hauptinhalt geben dem Look den letzten Schliff.

Das Stylesheet, das wir bisher erstellt haben, finden Sie in Listing 7.2.

Listing 7.2 Die bisher erstellten Styles

```
html, body {margin: 0; padding: 0;
  color: #000; background: #EEF;
  font-family: Verdana, Arial, sans-serif;}
h1 {color: #FFF; background: rgb(0%,56%,84%);
  font: bold 200%/1em Arial, Verdana, sans-serif;
  padding: 1em 1em 0; margin: 0;
  border: 1px solid rgb(0%,31%,46%);
  border-width: 2px 0;}
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%; clear: left;
  border-top: 1px solid gray;}
#nav {margin: 0; padding: 0;}
#nav li {list-style: none; float: left; margin: 0; width: 15%;
font-size: 10px; line-height: 20px; white-space: nowrap;}
#nav a {display: block; float: left; padding: 0 0 0 20px;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #CCC url(tab-bumps.gif) 2px 50% no-repeat;
  color: #333;}
/* Kommentierter Backslash Hack versteckt Regel vor dem IE5/Mac \*/
#nav a {float: none;}            /* Ende IE5/Mac Hack */
#nav a:hover {background-color: #333; color: #CCC;
  border-color: #000 #CCC #CCC #000;}
```

7.4.5 Beschnittener Text und geschrumpfte Hotspots

Jetzt ist ein passender Moment, um uns um zwei potenzielle Probleme kümmern, die bei den bisher von uns geschriebenen Styles auftreten können. Dabei geht um Folgendes:

- Wenn der Text eines Links zu lang ist, ragt er aus der Link-Box hervor und kann durch andere Inhalte abgeschnitten werden. Das haben wir zuerst in Abbildung 7.11 gesehen.
- Im IE/Win ist der »Hotspot« (der anklickbare Bereich) für jeden Link auf den eigentlichen Inhalt beschränkt und füllt nicht die ganze Box des Listenelements aus.

Wenden wir uns dem ersten Problem zu. Der Text wird abgeschnitten, weil wir mit `white-space: nowrap` verhindert haben, dass der Text umgebrochen wird, und weil wir den Listenelementen eine explizite `width` von `15%` gegeben haben. Somit müssen die Listenelemente alle 15 % so breit sein wie ihr jeweiliges Elternelement (das `ul`-Element), egal ob ihr Inhalt tatsächlich darin genug Platz hat oder nicht. Das Gleiche gilt

Auswüchse

Tatsächlich fließt der Text aus dem Listenelement heraus und wäre außerhalb des Listenelements sichtbar, wären da nicht die anderen Floats, die den herausfließenden Text überschreiben. Darum ist der Text nicht wirklich abgeschnitten, obwohl es in unseren Projektdateien so aussieht. Damit Sie sehen können, was wirklich passiert, versuchen Sie einfach mal, alles bis auf den ersten Link im Navigationsbereich auszukommentieren.

für jede explizit festgelegte Breite, egal ob sie in Prozent, Pixel, ems oder einer anderen Längenangabe ist.

Wir könnten einen Textumbruch zulassen, indem wir die `white-space`-Deklaration entfernen, aber dann wäre der Text bei einigen Links umgebrochen worden und bei anderen nicht, was zu ungleichen Höhen bei den Links führt. Diesen Effekt wollen wir definitiv nicht.

Nun wollen wir uns dem zweiten Problem widmen, dessen Lösung uns auch bei der Beseitigung des ersten Problems helfen wird. Beim IE/Win verhindert ein Bug, dass die Gesamtheit eines Blockbox-Links als »anklickbar« erkannt wird. Wenn nicht irgendeine Art von Breite explizit angegeben wird, wird nur der Inhaltsbereich des Links als anklickbar betrachtet. Wir könnten die Links so einstellen, dass sie eine `width` von `100%` haben, außer dass wir dann den Links kein Padding mehr hinzufügen könnten, ohne dass sie aus den Listenelementen herausragen (weil `width` die Breite des Inhaltsbereichs plus Padding, Rahmen und Ränder definiert). In diesem Fall könnten wir den Linkhintergründen keine Bilder hinzufügen, was uns zu sehr einschränkt.

Um diese Probleme anzupacken, hat Doug Bowman eine Lösung erarbeitet, die einen anderen Bug im IE/Win ausnutzt, um den Browser dazu zu verleiten anzunehmen, dass der ganze Link anklickbar ist. Wenn bei den Links für `width` ein winziger Wert angegeben wird, dann erweitert der IE/Win automatisch (und unkorrekterweise) den Wert, um den gesamten Inhalt mit einzuschließen, *und* er geht obendrein davon aus, dass das Padding anklickbar ist. Das ist schon ziemlich sonderbar, aber es funktioniert. Also dann:

```
#nav a {display: block; float: left; padding: 0 0 0 20px;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #CCC url(tab-bumps.gif) 2px 50% no-repeat;
  color: #333;
  width: .1em;}
```

Das Problem dabei ist, dass konformere Browser den Wert ernst nehmen und den Bereich für den Linkinhalt genau ein Zehntel eines em breit machen, und der Text fließt dann aus diesem schmalen Fitzelchen heraus. Also müssen wir den Schaden für diese fortgeschrittenen Browser ungeschehen machen, indem wir ihnen eine Regel zeigen, die der IE/Win nicht sieht. In diesem Fall verlassen wir uns auf die Tatsache, dass der IE/Win keine Kind-Selektoren versteht, und fügen eine Regel hinzu, die für fortschrittlichere Browser die `width` wieder zurück auf `auto` setzt.

```
#nav a {display: block; float: left; padding: 0 0 0 20px;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #CCC url(tab-bumps.gif) 2px 50% no-repeat;
  color: #333;
  width: .1em;}
html>body #nav a {width: auto;}    /* fixes IE6 hack */
/* Kommentierter Backslash Hack versteckt Regel vor dem IE5/Mac \*/
```

So werden alle Browser sehen, dass die Breite `.1em` sein soll, und die, die Kind-Selektoren verstehen, werden auch `width:auto` sehen und es verwenden, um den Wert `.1em` zu überschreiben. Auf beide Arten wird dank der Art, wie Floats sich genau ihren Inhalten anpassen, die Breite des Linkinhalts exakt gleich der Breite des Textinhalts sein, und der ganze Link wird anklickbar – siehe Abbildung 7.13.

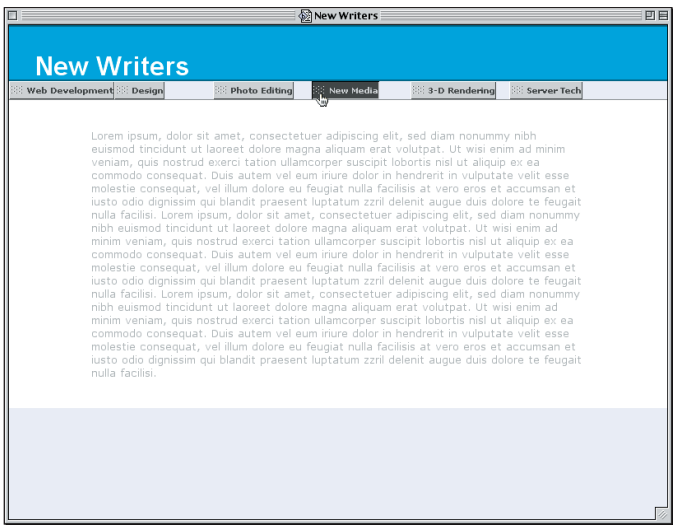


ABBILDUNG 7.13

Die Links schrumpfen so, dass sie sich ihren Inhalten anpassen.

Ach du liebe Güte – was ist denn jetzt passiert? Genau, was wir beschrieben haben: Die Inhaltsbereiche der Links werden so breit wie der Inhaltstext. Die Listenelemente sind aber immer noch 15 % breit, also füllen die Links die Listenelemente nicht mehr aus.

Das passiert also zumindest in einigen Browsern. (Abbildung 7.13 wurde mit einem IE5/Mac gemacht, und beim IE/Win sollte es genauso aussehen.) Bei anderen werden die Links immer noch die Listenelemente ausfüllen. Das ist nun eindeutig nicht viel besser als vorher, als es noch unser einziges Problem war, dass bei einem der Links der Text aus dem Button herausragte.

Wir können alles wieder auf harmonische Art einrichten, wenn wir die Links so schrumpfen lassen können, dass sie sich ihren Inhalten anpassen. Dafür müssen wir den Listenelementen für die Breite den Wert `auto` geben, und das machen wir, indem wir einfach aus der Regel `#nav li` den Wert `width: 15%` entfernen.

```
#nav li {list-style: none; float: left;
  font-size: 10px; line-height: 20px; white-space: nowrap;}
```

Jetzt wird bei jedem Button der Link so schrumpfen, dass er zum Inhalt passt, und das Listenelement wird auf den Link zusammenschrumpfen. Das führt uns zu der in Abbildung 7.14 gezeigten Situation.

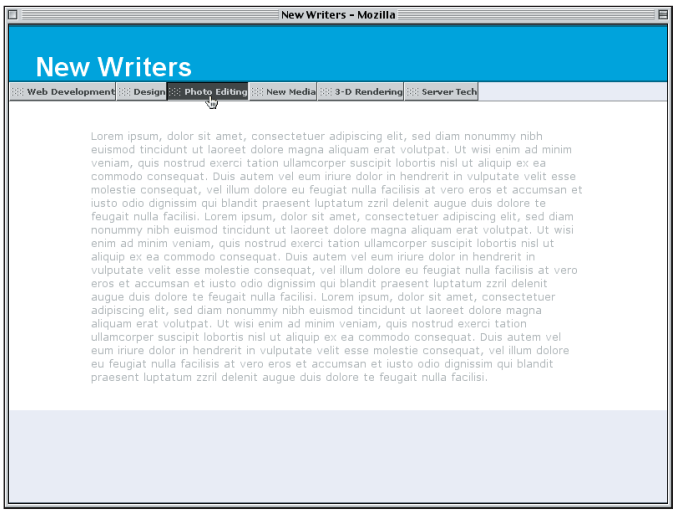


ABBILDUNG 7.14

Sowohl die Listenelemente als auch die Links werden auf passende Größe geschrumpft.

Was haben wir bei diesem kleinen Zwischenspiel gelernt?

- Dass es schwierig ist, im IE/Win vollständig anklickbare Links zu bekommen, wenn ihre Elternelemente auf eine explizite Breite einstellt sind.
- Dass bei auf eine explizite Breite eingestellten Links das Risiko besteht, dass der Inhalt aus den Links herausfließt.
- Dass beide Probleme durch passende Schrumpfung der Listenelemente und der darin enthaltenen Links gelöst werden können, nachdem Sie einige Bugs im Internet Explorer umgangen haben.

Also kommen wir zu dem vernünftigen Schluss, dass es im Allgemeinen besser ist, sich selbst in der Größe anpassende Links zu haben als solche mit expliziten Größenangaben. Und das ist auch der Ansatz, der in der »Sliding Doors«-Technik angewendet wird: Tabs, die sich in der Größe von selbst ihren Inhalten anpassen.

Und es gibt noch weitere Gründe, bei dieser Technik das Schrumpfen einzusetzen, wie wir bald herausfinden werden.

7.5 Erstellung der Tabs

Obwohl die von uns kreierten Link-Styles auf ihre eigene Art ganz schön sind, haben sie etwas mit den meisten Links heutzutage gemeinsam. Sie sind schlicht rechteckig und irgendwie langweilig. Wir brauchen da andere, attraktivere Link-Styles – mit abgerundeten Ecken zum Beispiel, mit einer dezenten Tönung und visuellen Akzentuierung.

Wie Sie bereits erraten haben, ist das machbar, und dafür ist auch keine Änderung des HTML-Markups nötig. Wir brauchen nur etwas modifiziertes CSS und ein großes Bild (jedoch nicht auf die Dateigröße bezogen).

7.5.1 Ein paar Änderungen

Bevor wir uns an den schönen Teil der Arbeit machen, wollen wir bei dem Stylesheet aus dem Listing 7.2 einige Sachen ändern. Die erste Änderung stellt den Hintergrund des Dokuments von dem blassen Graublau auf Weiß um.

```
html, body {margin: 0; padding: 0;
  color: #000; background: #FFF;
  font-family: Verdana, Arial, sans-serif;}
```

Als Nächstes werden wir den grauen Rahmen vom `div`-Elements des Hauptinhalts oben entfernen, so dass die `#main`-Regel wie folgt aussieht:

```
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%; clear: left;}
```

Schließlich werden wir den gefloateten Listenelementen einen linken Rand und etwas Padding links geben. Das wird sie ein bisschen auseinander schieben und Platz zwischen der linken Kante der Listenelemente und der linken Kante der Links schaffen, die sie enthalten (siehe Abbildung 7.15).

```
#nav li {list-style: none; float: left;
  margin-left: 1px; padding-left: 16px;
  font-size: 10px; line-height: 20px; white-space: nowrap;}
```

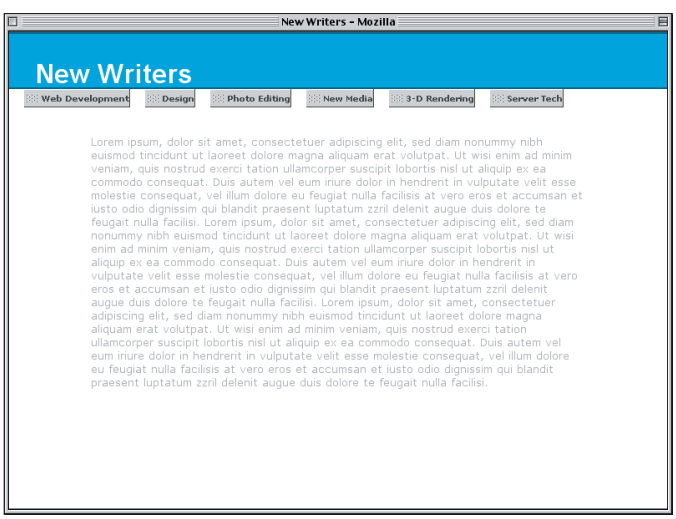


Abbildung 7.15

Auseinander gezogene Links bereiten das fortgeschrittene Styling vor.

Und was sollte das jetzt? Damit schaffen wir Platz, um einen kleinen Blickfang für das Auge zu setzen.

7.5.2 Ein Tab wird eingesetzt

Unser prinzipielles Ziel hier ist, sowohl bei den Listenelementen als auch den Links ein visuell ansprechendes Bild in den Hintergrund zu legen. Wir werden diese Hintergründe so stylen, dass sie nahtlos ineinander verschmelzen und bei Links unterschiedlicher Breite gut aussehen.

Damit dieser Effekt funktioniert, brauchen wir zuerst ein passendes Bild für den gewünschten Effekt. Tatsächlich brauchen wir ein ziemlich großes Bild ... ein Bild wie das in Abbildung 7.16 mit dem Namen `tabs2-big.gif`.

ABBILDUNG 7.16
Das Mega-Tab-Bild

Achten Sie auf die abgerundeten Ecken am unteren Rand des Bildes. Diese repräsentieren die beiden Seiten unserer Tabs, die aus dem Seitentitel nach unten ragen werden.

Das wollen wir nun Schritt für Schritt durchführen. Zuerst werden wir dieses Bild des Tabs bei den Listenelementen einfügen.

```
#nav li {list-style: none; float: left;
  margin-left: 1px; padding-left: 16px;
  font-size: 10px; line-height: 20px; white-space: nowrap;
  background: #BBB url(tabs2-big.gif);}
```

Denken Sie daran, dass wir den Listenelementen 16 Pixel Padding links gegeben haben, also wird das Bild in diesem Bereich erscheinen. Der Rest jedes Listenelements wird durch die Links gefüllt, die einen opaken Hintergrund haben und darum das Bild verdecken, das im Hintergrund der Listenelemente steht.

Was wir als Ergebnis sehen, ist, dass die obere linke Ecke von `tabs2-big.gif` an der oberen linken Ecke der Listenelemente ausgerichtet ist. Wir wollen, dass die untere linke Ecke des Bildes in der gleichen Ecke wie das Listenelement ist, und aus diesem Grunde soll das Bild nicht als Kachel dargestellt werden. Wenn wir der Hintergrund-

Ein Bild oder zwei?

In der ursprünglichen Technik hat Doug dieses Bild in zwei ungleiche Hälften geteilt, anstatt es bei einem einzelnen Bild zu belassen. Der Grund dafür ist, dass er Bereiche des Hintergrunds auf transparent gestellt hat, und das macht die Teilung erforderlich (in seinem Artikel erfahren Sie weitere Details). Da wir keine Transparenz verwenden, bleibt es bei einem einzelnen Bild.

Deklaration noch ein bisschen hinzugeben, reicht das aus, damit wir uns um diese Punkte nicht weiter kümmern müssen, und das Ergebnis sehen Sie in Abbildung 7.17.

```
#nav li {list-style: none; float: left;
  margin-left: 1px; padding-left: 16px;
  font-size: 10px; line-height: 20px; white-space: nowrap;
  background: #BBB url(tabs2-big.gif) 0 100% no-repeat;}
```

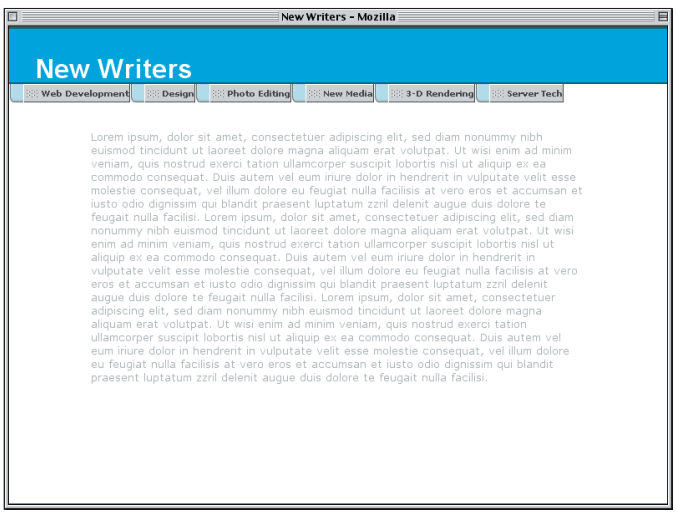


ABBILDUNG 7.17

Das Mega-Tab-Bild wird im Hintergrund des Listenelements platziert.

Das ist schon für sich genommen ein interessanter visueller Effekt bzw. hat das Zeug dazu. Wir wollen das aber noch weiter vorantreiben. Damit die Tabs zusammenrücken, müssen wir `tabs2-big.gif` auf die Links anwenden, nur werden wir dieses Mal die untere rechte Ecke des Bildes an der unteren rechten Ecke der Links ausrichten.

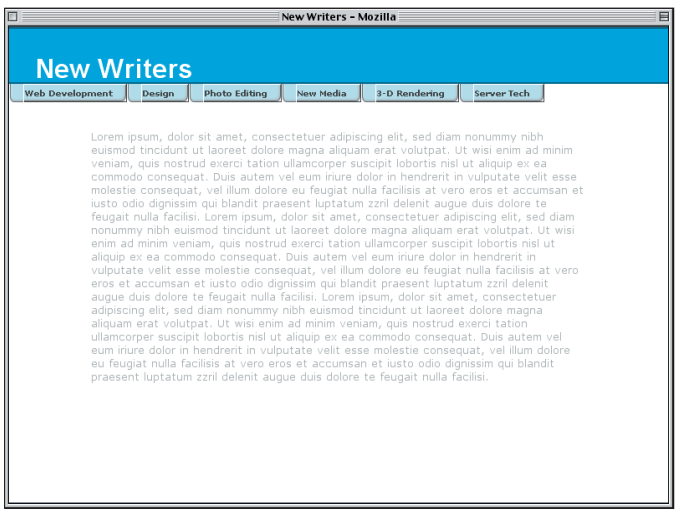
```
#nav a {display: block; float: left; padding: 0 0 0 20px;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #DDD url(tabs2-big.gif) 100% 100% no-repeat;
  color: #333;
  width: .1em;}
```

Es gibt noch eine andere Änderung, die durch den Wechsel des Hintergrundbildes erforderlich wird. Denken Sie daran, dass wir vorher ein Hintergrundbild links hatten und etwas Padding hinzugefügt haben, um einen Bereich freizumachen, in dem es erscheinen konnte. Nun haben wir das Hintergrundbild nach rechts ausgerichtet. Also müssen wir das Padding der Links entsprechend anpassen. Wir werden das linke Padding entfernen und 16 Pixel Padding rechts hinzufügen (damit das dem `padding-left: 16px` bei den Listenelementen entspricht), und das Ergebnis sehen Sie in Abbildung 7.18.

```
#nav a {display: block; float: left; padding: 0 16px 0 0;
  text-decoration: none; font-weight: bold;
  border: 1px solid; border-color: #FFF #333 #333 #FFF;
  background: #DDD url(tabs2-big.gif) 100% 100% no-repeat;
  color: #333;
  width: .1em;}
```

ABBILDUNG 7.18

Das Mega-Tab-Bild wird im Hintergrund der Links platziert.



Wir sind schon fast fertig, aber da gibt es noch die kleine Sache mit den Rahmen der Links. Sie werden nicht länger gebraucht; vielmehr wirken sie sich sogar negativ auf den Effekt aus, den wir schaffen wollen. Also müssen wir die Rahmen komplett entfernen, und damit sieht die Regel `#nav a` wie folgt aus:

```
#nav a {display: block; float: left; padding: 0 16px 0 0;
  text-decoration: none; font-weight: bold;
  background: #DDD url(tabs2-big.gif) 100% 100% no-repeat;
  color: #333;
  width: .1em;}
```

Wir müssen auch die Rollover-Styles anpassen, die bei den hellblauen Tabs nicht sonderlich funktional sind. Da wir bloß die Links manipulieren können, werden wir einfach deren Textfarbe ändern.

```
#nav a:hover {color: rgb(62%,35%,22%);}
```

Wir könnten auch das Hintergrundbild ändern, aber dann hätten wir nur einen geänderten Link-Hintergrund. Der Hintergrund des Listenelements würde sich nicht ändern. Darum ist es ganz schön schwierig, in einer Situation wie dieser Rollover-Effekte auf etwas anderes als den Link anzuwenden. Zum Glück reicht das meist aus, und das können wir uns in Abbildung 7.19 ansehen.

Abbildung 7.19

Die Links erhalten einen Rollover-Effekt.

Bessere Rollover-Effekte

Es ist in der Tat möglich, in den meisten modernen Browsern Rollover-Effekte bei Elementen zu erzielen, die keine Links sind. Das könnten wir nutzen, um Rollover-Effekte für die von uns erstellten Tabs einzubauen. Nehmen wir beispielsweise an, wir haben ein »Hervorhebungs«-Bild namens `tabs2-big-hl.gif`*. Von der visuellen Erscheinung her ist es die gleiche Datei wie* `tabs2-big.gif`*, nur dass die Farben leuchtender sind.*

Wir könnten mit den folgenden Regeln einen Rollover-Effekt zum Hervorheben schaffen:

```
#nav li:hover, #nav li:hover a {
  background-image: url(tabs2-big-hl.gif);}
```

Damit wird sowohl das Hintergrundbild des Listenelements, über dem der Mauszeiger ruht, als auch das Hintergrundbild des Links darin durch das »Hervorhebungs«-Bild ersetzt.

Der Nachteil bei diesem Ansatz ist, dass der Explorer diese Regel völlig ignorieren wird, und das heißt, für User mit diesem Browser gibt es keinen Rollover-Effekt. User mit anderen Browsern wie Mozilla, Opera und Safari erhalten dagegen eine Hervorhebung.

Damit sogar der Explorer Rollover-Effekte anwendet, gibt es zwei Wege. Bei einem wird ein Span-Element in die Links eingefügt und gemeinsam mit dem Link gestylt. Schauen Sie sich »Sliding Doors of CSS, Part II« an, dort finden Sie ein Beispiel, das diese Technik illustriert (`http://alistapart.com/articles/slidingdoors2/`*). Beim anderen Weg benutzen wir die Behavior-Datei für den IE/Win aus Projekt 6.*

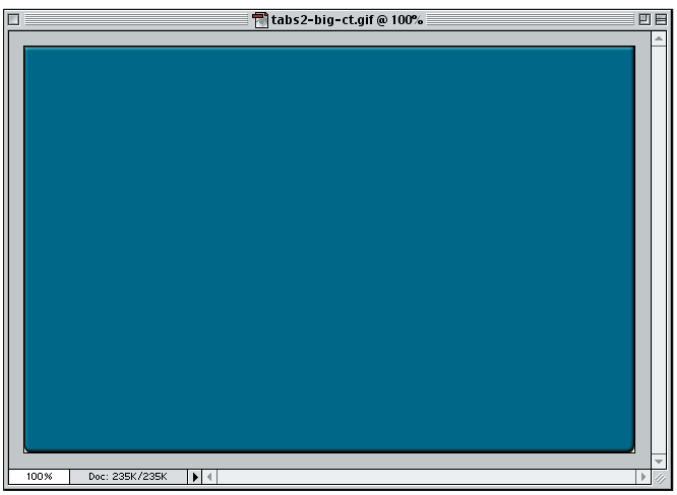
7.5.3 Anzeige des aktuellen Tabs

Erinnern Sie sich an `id="current"`, das wir seit Beginn dieses Kapitels ignoriert haben? Nun kommt es mit Donnerhall zurück. Über diesen Hook können wir den Tab ganz unverwechselbar stylen, so dass er sich von den anderen unterscheidet.

Dafür brauchen wir allerdings ein neues Tab-Bild, eines, das anders als das normale Tab-Bild aussieht. Und Abbildung 7.20 zeigt genau das.

ABBILDUNG 7.20

Das Mega-Tab-Bild für die aktuelle Seite



Der Name der Datei lautet `tabs2-big-ct.gif` (`ct` für »*current tab*« - aktueller Tab). Die Grundfarbe des Tabs wurde den Rahmen entnommen, die auf dem Seitentitel oben und unten verlaufen, die übrigens `rgb(0%,31%,46%)`ist – aus Gründen, die wir gleich sehen werden.

Nun, wo dieses Bild einsatzbereit ist, brauchen wir nur eine Regel zu schreiben, die das Basis-Tab-Bild durch das neue ersetzt und die Textfarbe ändert. Richtig erkannt – wir brauchen nur eine Regel.

```
#nav a:hover {color: rgb(62%,35%,22%);}
#nav #current, #nav #current a {color: #FDB;
  background-image: url(tabs2-big-ct.gif);}
</style>
```

Damit wird das Bild für den aktuellen Tab durch das Standard-Tab-Bild ersetzet, während die Styles für dessen Position, Wiederholung usw. intakt bleiben. Die Farbe wird konstant bleiben, auch wenn der Mauszeiger über dem Tab steht, weil der Selektor `#nav #current a` über eine höhere Spezifität verfügt als `#nav a:hover` (dass er nach der Rollover-Regel kommt, ist in diesem Fall irrelevant).

Wie wir in Abbildung 7.21 sehen können, wirkt es so, als ob der Tab aus dem Rahmen hervorsticht, wobei die visuelle Konsistenz mit den anderen Tabs immer noch erhalten bleibt.

Abbildung 7.21

Das Erscheinen des aktuellen Tabs wird verändert.

So können wir es schaffen, nur mit einer unsortierten Liste und ein paar Links dem Layout einige attraktiv wirkende Tabs zu geben. Das neue Stylesheet finden Sie in Listing 7.3.

Listing 7.3 Die Styles der Tabs

```
html, body {margin: 0; padding: 0;
  color: #000; background: #FFF;
  font-family: Verdana, Arial, sans-serif;}
h1 {color: #FFF; background: rgb(0%,56%,84%);
  font: bold 200%/1em Arial, Verdana, sans-serif;
  padding: 1em 1em 0; margin: 0;
  border: 1px solid rgb(0%,31%,46%);
  border-width: 2px 0;}
#main {font-size: small; color: #AAA; background: #FFF;
  margin: 0; padding: 2.5% 12.5%; clear: left;}
#nav {margin: 0; padding: 0;}
#nav li {list-style: none; float: left;
  margin-left: 1px; padding-left: 16px;
  font-size: 10px; line-height: 20px; white-space: nowrap;
  background: #BBB url(tabs2-big.gif) 0 100% no-repeat;}
#nav a {display: block; float: left; padding: 0 16px 0 0;
  text-decoration: none; font-weight: bold;
  background: #DDD url(tabs2-big.gif) 100% 100% no-repeat;
  color: #333;
  width: .1em;}
html>body #nav a {width: auto;}    /* fixes IE6 hack */
/* Kommentierter Backslash Hack versteckt Regel vor dem IE5-Mac \*/
#nav a {float: none;}            /* End IE5-Mac hack */
#nav a:hover {color: rgb(62%,35%,22%);}
#nav #current, #nav #current a {color: #FDB;
  background-image: url(tabs2-big-ct.gif);}
```

7.5.4 Was noch zu sagen bleibt

Falls Ihnen die zusätzliche Seitengröße Kopfschmerzen bereitet, da wir nur für ein paar Tabs zwei große Bilder verwendet haben, können Sie unbesorgt sein. Die beiden Bilder sind weniger als 3 KB groß, und die Dateigröße des normalen Tabs ist etwas geringer (2.763 Bytes). Das heißt, dass durch die Nutzung dieser speziellen Technik der Seitengröße weniger als 6 KB hinzugefügt wurden.

Natürlich braucht man auch keine 600 x 400 Pixel großen Bilder zu nehmen, also könnten die Dateien auch kleiner sein. Alternativ ist es möglich, dass die Erscheinungsbilder des aktuellen Tabs und des normalen Tabs in einem einzigen Bild kombiniert werden und `background-position` verwendet wird, um den entsprechenden Effekt zu erhalten. Diese Technik werden wir hier nicht ausprobieren, weil Doug Bowman sie so vorzüglich in seinem Artikel »Sliding Doors of CSS, Part II« vorgestellt hat, und den sollten Sie definitiv lesen (zusammen mit seinem Vorläufer, in dem ebenfalls gezeigt wird, wie Tabs visuell mit dem Bereich des Hauptinhalts verschmelzen können).

Ein weiterer Punkt ist, dass es wirklich leicht ist, das Erscheinungsbild Ihrer Tabs zu verändern – Sie brauchen bloß das Mega-Tab-Bild zu aktualisieren, und das war's. (Wenn Sie die Ausrichtung der Tabs ändern wollen, müssen Sie das CSS auch aktualisieren). Nehmen Sie beispielsweise die Tabs aus der Abbildung 7.22, die unter Verwendung des darunter gezeigten Mega-Tab-Bildes erstellt wurden.

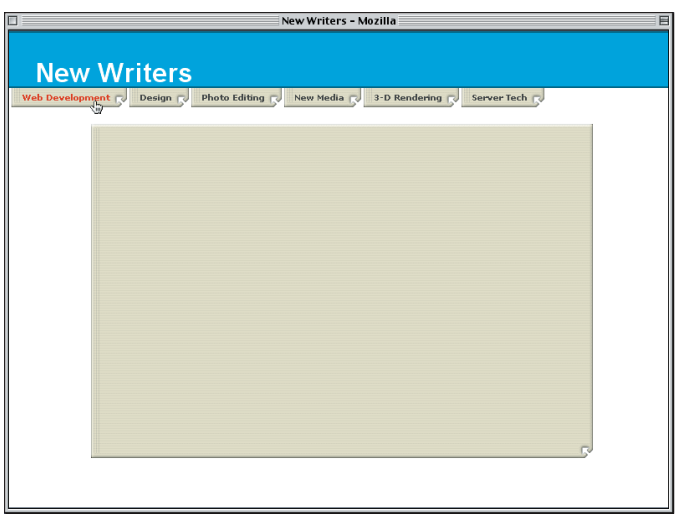


ABBILDUNG 7.22
Radikale Änderung der Tabs über einen Austausch des Mega-Tabs

Das bringt uns zum letzten Punkt: den verschiedenen visuellen Styles, die über die »Sliding Doors«-Technik kreiert werden können. Im Allgemeinen wird jeder Tab, der eine solide Hintergrundfarbe verwendet oder horizontal ausgerichtete Dekorationen einsetzt, ganz prima funktionieren. Beispielsweise laufen bei den Tabs aus Abbildung 7.22 horizontale Linien durch den Hintergrund. Die vertikalen Linien auf der linken

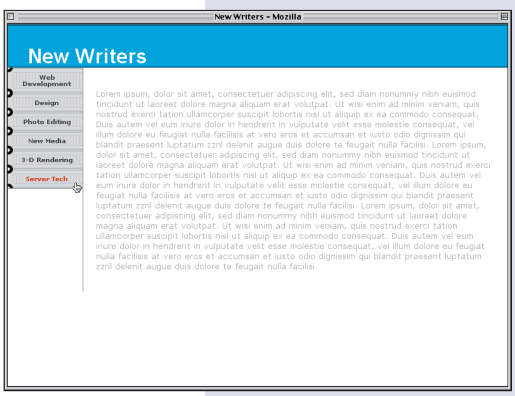
Seite wurden vorsichtig in dem Bereich platziert, der in dem Hintergrund des Listenelements erscheint; das heißt, die vertikalen Linien stoppen vor der linken Kante des Links.

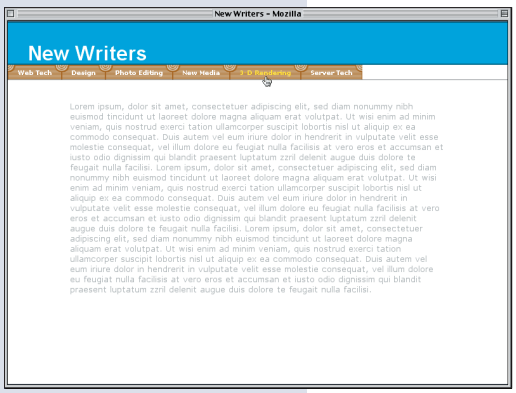
Entsprechend ist der Umgang mit diagonalen Linien ziemlich schwer. Das Problem ist, dass dort, wo die »Türen« zusammen kommen, die diagonalen Linien vielleicht nicht sauber aufeinander treffen. Das heißt nicht, dass in Tabs niemals diagonale Linien verwendet werden können. Es bedeutet nur, dass Sie visuell kreativer sein müssen, um die gerade erwähnten Einschränkungen überwinden zu können.

7.6 Spielwiese

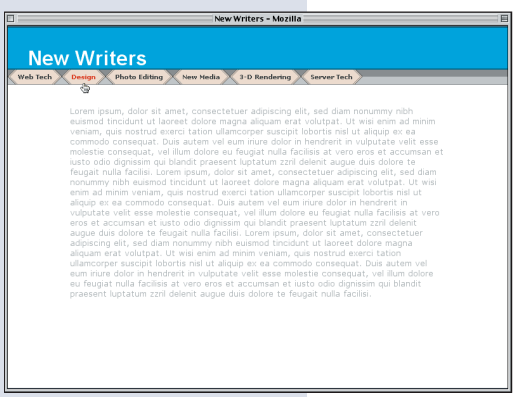
Da wir nun gesehen haben, wie mit einfachem Markup Buttons und Tabs erstellt werden können, sollen Sie Ihre Chance bekommen, diese Konzepte noch weiter zu erforschen.

1. Versuchen Sie, die »Sliding Doors«-Technik auf eine vertikale Linkliste anzuwenden, die so hoch oder kurz sein soll, wie es Ihnen gefällt. Beachten Sie, dass es in diesem Fall keine große Sache ist, wenn die Links auf zwei oder mehr Zeilen umgebrochen werden.





2. Kreieren Sie Ihren eigenen Tab-Effekt und implementieren Sie ihn über das gleiche Markup und die gleichen Techniken, die wir in diesem Projekt ausprobiert haben. Je kreativer, desto besser!



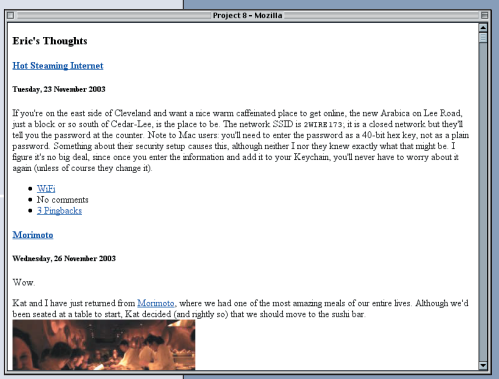
3. Als besondere Herausforderung können Sie versuchen, rhombenförmige Links mit einem gestreiften Hintergrund zu erstellen und dieses Thema über den ganzen »Navigationsbalken« durchzuhalten. Hinweis: Dazu brauchen Sie zwei Hintergrundbilder: eines für die Tabs und eines für das Element, das sie enthält.

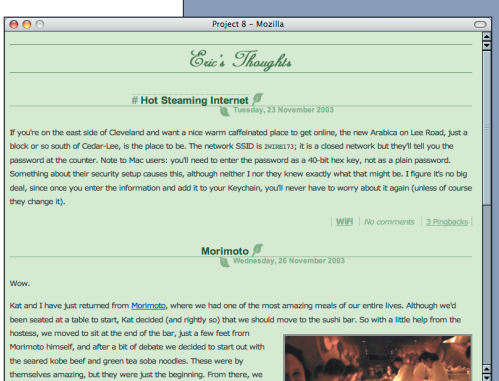
8

Ein Weblog wird gestylt

Paar ruhige Tage jetzt ... na, eigentlich nicht wirklich, aber ich werd's euch nicht erzählen. Und darum geht es ja auch bei einem Weblog.

Dave Whyte





Aus nicht ganz erklärlichen Gründen ist es dazu gekommen, dass persönliche, webbasierte Tagebücher als »Weblogs« bezeichnet werden (jedenfalls von den meisten Leuten). Wenn man es sich genau überlegt, ist ein Weblog schon ein recht interessanter Layout-Mikrokosmos. Jeder Eintrag in einem Weblog enthält gewöhnlich einen Titel, das Datum, an dem der Eintrag ins Netz gestellt wurde, den jeweiligen Inhalt und dann noch weitere Zusatzinformationen, z.B. über die Kategorie des Eintrags, einen Link zu weiteren Kommentaren usw.

Vom Standpunkt des Layouts her muss jeder Eintrag als eine Art Mini-Dokument innerhalb der größeren Seite betrachtet werden. Jeder Eintrag sollte auf die gleiche Weise wie andere Einträge gestylt werden und auf eine visuell ansprechende Weise mit den anderen verbunden sein. Es wäre beispielsweise nicht gut, wenn die Einträge überlappen. In diesem Projekt werden wir uns ein Weblog vornehmen, das auf einem sauberen und gut strukturierten Markup beruht, und ausprobieren, wie die Einträge gestylt werden können.

8.1 Projektziele

In diesem Projekt geht es nur darum, die Einträge in einem Weblog gut aussehen zu lassen. Während wir daran arbeiten, müssen wir darauf achten, den visuellen Zusammenhang zwischen den Einträgen zu wahren. Das führt uns zu den folgenden allgemeinen Zielen:

- Jeder Eintrag soll visuell unterscheidbar sein und für sich selbst stehen. Das bedeutet nicht, um jeden Eintrag einen Kasten zu ziehen – wir müssen nur sicher stellen, dass es eindeutig ist, wo ein Eintrag endet und der nächste beginnt.
- Wir werden darauf achten, dass der Titel und das Datum eines jeden Eintrags in sinnvoller Weise zusammenkommen, damit deutlich wird, dass sie in Beziehung zueinander stehen.
- Alle Zusatzinformationen sollen in ihrer visuellen Darstellung abgeschwächt werden, so dass sie nicht in Konkurrenz zum Haupteintrag stehen, aber auch nicht vollständig verschwinden.

Obendrein werden wir für das Design ein Thema aus der Natur wählen, das frisch grün und naturverbunden wirkt. Das hat nichts mit den vorangegangenen Punkten zu tun, sondern ist nur eine gleich am Anfang getroffene Designentscheidung.

8.2 Vorbereitungen

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

Laden Sie die Dateien für Projekt 8 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch08proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und neu laden.

8.3 Die Grundlagen

Wie sonst auch werden wir uns als Erstes in das Markup des Projektdokuments vertiefen, damit wir besser verstehen, womit wir es zu tun haben, und machen uns vor der Arbeit mit CSS das Dokument zu eigen. Im Listing 8.1 sehen wir das detaillierte Markup für das Weblog, und in Abbildung 8.1 finden wir es in seiner ungehobelten und ungestylten Herrlichkeit.

Listing 8.1 Ein Blick auf das Markup des Weblogs

```
<div id="weblog">

<h3><span>Eric's Thoughts</span></h3>

<div class="entry">

<h4 class="title">
<a name="t20031125" href="/eric/thoughts/200311.html#t20031123"
  rel="bookmark">Hot Steaming Internet</a>
</h4>
<h5 class="date">
Tuesday, 23 November 2003
</h5>

[...inhaltliche Einträge...]

<ul class="moreinfo">
<li class="categories"><a href="/eric/thoughts/wifi">WiFi</a></li>
<li class="comments">No comments</li>
<li class="pingbacks"><a href="/eric/thoughts/pingbacks/t20031123">3
 Pingbacks</a></li>
</ul>

</div>

[...weitere Einträge...]

</div>
```

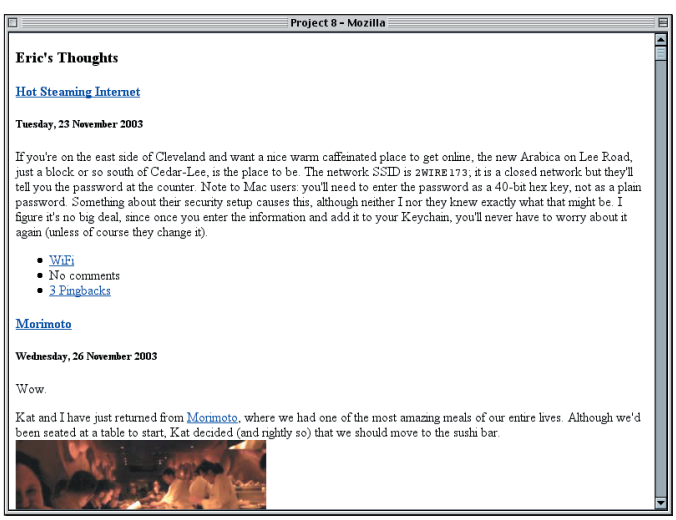


Abbildung 8.1

Die ungestylten Weblog-Einträge

Da gibt es ein paar Sachen, die extra angemerkt werden sollen:

- Der Titel des Weblogs (»Eric's Thoughts«) ist in einem `h3`- und einem `span`-Element enthalten. Damit haben wir zwei Elemente, mit denen wir beim Styling des Titels arbeiten können.
- Jeder Eintrag wird von einem `div`-Element mit einer `class` namens `entry` umschlossen. Damit wird die gesamte Information, die zu einem einzelnen Eintrag gehört, sauber in einem einzigen Teil der Dokumentstruktur gruppiert.
- Der Titel und auch das Datum eines Postings, denen eindeutige Elemente für den Inhalt gegeben wurden (`h4` bzw. `h5`), wurden ebenfalls mit einer Klasse versehen. Das erlaubt uns, diese Aspekte eines Eintrags zu stylen, ohne uns darum sorgen zu müssen, dass eines Tages vielleicht mal ein `h4` oder `h5` beim Haupttext eines Eintrags benutzt werden könnte.
- In ähnlicher Weise sind die Zusatzinformationen, die in einer unsortierten Liste enthalten sind, einer Klasse zugeordnet worden, über die man leicht jedes Teil unabhängig von allen anderen stylen kann und ohne dass die Styles in den Haupttext eines Eintrages sickern könnten.

Wir werden uns alle diese Punkte zunutze machen und auch noch eine Vielzahl anderer Dinge, um diese Einträge besser als je zuvor aussehen lassen.

8.4 Weblog-Styling

Ein zweckmäßiger erster Schritt besteht darin, die Grundlinien für die Hintergrundfarbe, die Textfarbe und Schriftfamilie und -größe für das Gesamtdokument zu bestimmen. Wir werden es auf ein Naturthema hin auslegen, also werden wir uns bei den Farben für verschiedene Grüntöne entscheiden. Der Font wird, wie es bei diesen persönlichen Sites so oft der Fall ist, einen Sans-Serif-Font sein, der kleiner als der Standard ist,.

```
<style type="text/css">
body {color: rgb(18%,19%,17%); background: rgb(85%,92%,81%);
 font: 0.85em Verdana, sans-serif;}
</style>
```

Das haben wir nun passend gemacht, und wir können uns nun Stück für Stück durch den HTML-Code arbeiten und die Teile der Reihe nach stylen, wie sie uns begegnen. Damit steht also der Titel des Weblogs gleich am Anfang der Reihe.

8.4.1 Nun zum Titel

Unsere erste »Baustelle« ist »Eric's Thoughts«, der Titel des Weblogs. Die wohl einfachsten Sachen sind, die Schrift zu vergrößern und den Text zu zentrieren.

```
body {color: rgb(18%,19%,17%); background: rgb(85%,92%,81%);
 font: 0.85em Verdana, sans-serif;}
#weblog h3 {font-size: 150%; text-align: center;}
</style>
```

Das wird für sich genommen den Titel allerdings kaum vom Rest des Dokuments abheben. Um ihm eine bessere visuelle Ausgestaltung zu geben, werden wir den Text sowohl farbig machen als auch oben und unten einen Rahmen und zusätzlich einen unteren Rand, der etwas breiter als gewöhnlich ist, einfügen. Damit wird der Rest des Inhalts ein wenig nach unten geschoben – siehe Abbildung 8.2.

```
#weblog h3 {font-size: 150%; text-align: center;
  color: rgb(10%,30%,10%);
  border: 1px solid rgb(30%,50%,30%); border-width: 1px 0;
  margin-bottom: 1.5em;}
```

Okay, der Titel sticht hervor ... aber wirkt er ansprechend? Nein, nicht wirklich. Er sähe mit einem Handschriften-Font wirklich besser aus. CSS erlaubt als generischen Schriftfamilientyp `cursive`, der aber nicht sonderlich gut von den Browsern unterstützt wird. Wenn ein User den eigenen Browser so konfiguriert hat, dass er spezielle Handschriften-Fonts erkennt, ist alles klar. Aber die meisten haben das nicht. Wenn wir also `font-family: cursive;` eingeben, würden die meisten User einfach nur einen Serif- oder Sans-Serif-Font bekommen.

ems oder Prozent?

Wenn es um die Größenberechnung von Fonts geht, gibt es zwischen em und Prozentwerten theoretisch keinen Unterschied. `150%` und `1,5em` führen zum gleichen Ergebnis, genau wie `0,85em` und `85%`. Allerdings wird meist empfohlen, dass wenn Sie die Schriftgröße des `body`-Elements auf `1em` setzen, Sie diesen Wert anstelle von `100%` verwenden sollten, denn sonst kann es zu einem merkwürdigen Rundungsfehler bei Opera kommen.

ABBILDUNG 8.2

Die Auswirkungen der ersten einfachen Styles

Also entscheiden wir uns für ein Bild statt für Text. Diese Technik wird auch als *Image Replacement* (Bild-Ersetzung) bezeichnet. Wir werden also ein Bild nehmen (genaugenommen das aus Abbildung 8.3) und es in den Hintergrund von `h3` legen, während wir den Text komplett aus dem Weg räumen.

Abbildung 8.3

Der Titel des Weblogs als Bild

Zuerst setzen wir das Bild in den Hintergrund von `h3`, positionieren es in dessen unterer Mitte und achten darauf, dass es nur einmal erscheint.

```
#weblog h3 {font-size: 150%; text-align: center;
  color: rgb(10%,30%,10%);
  border: 1px solid rgb(30%,50%,30%); border-width: 1px 0;
  margin-bottom: 1.5em;
  background: url(title.gif) 50% 100% no-repeat;}
```

Nun sitzt das Hintergrundbild genau unter dem Text im Vordergrund, was in einigen Situationen ein schöner visueller Effekt sein kann, aber bei dieser nicht. Der Text darf wirklich gar nicht mehr im Weg stehen. Leichter getan als gesagt: Wir nehmen einfach das `span`-Element und kicken es ganz aus dem Bildschirm. Obwohl die Farbe des `h3` nicht mehr relevant ist, weil der Text nicht zu sehen ist, werden wir sie doch noch am Platz lassen.

```
#weblog h3 {font-size: 150%; text-align: center;
  color: rgb(10%,30%,10%);
  border: 1px solid rgb(30%,50%,30%); border-width: 1px 0;
  margin-bottom: 1.5em;
  background: url(title.gif) 50% 100% no-repeat;}
#weblog h3 span {position: absolute; left: -50em; width: 50em;}
</style>
```

Container mit root

In einem HTML-Dokument ist das root-Element das `html`-Element (nicht wirklich überraschend).

Jetzt ist das `span`-Element mit dem Text darin so platziert, dass sich die linke Kante des `span`-Elements 50em links von seinem Container-Block befindet. In diesem Fall ist der Container-Block das root-Element, was bedeutet, dass das `span`-Element 50 em links von der linken Kante des Browserfensters ist. Wir haben auch die `width` des `span`-Elements dem Offset-Abstand angeglichen, falls das `h3` aus irgendeinem Grund mal derart geändert wird, dass dort besonders viel Text eingetragen werden kann. Damit wäre der Text dann also versorgt.

Allerdings ist da noch ein Nebeneffekt, dem wir gegensteuern müssen. Durch die absolute Positionierung des `span`-Elements haben wir es aus dem normalen Fluss entfernt. Das bedeutet, das `h3` hat keinen normal fließenden Inhalt und besitzt somit keine Höhe. Wenn es keine Höhe aufweist, werden die oberen und unteren Rahmen direkt

aneinander stoßen, und wir kriegen den Hintergrund überhaupt nicht zu Gesicht. Um das zu reparieren, müssen wir bei `h3` eine explizite Höhe erzwingen, auch wenn kein Inhalt darin ist.

Das können wir leicht mit einer Höhen-Deklaration erreichen. Das Hintergrundbild ist 38 Pixel hoch, also sollte diese Zahl plus etwas Zugabe (damit der obere Rahmen ein paar Pixel Abstand vom Hintergrundbild hält) reichen. Wir wählen also 45 Pixel und bekommen als Ergebnis die Abbildung 8.4.

```
#weblog h3 {font-size: 150%; text-align: center;
  color: rgb(10%,30%,10%);
  border: 1px solid rgb(30%,50%,30%); border-width: 1px 0;
  margin-bottom: 1.5em; height: 45px;
  background: url(title.gif) 50% 100% no-repeat;}
```

ABBILDUNG 8.4

Das Aussehen des Titels wird über Image Replacement verbessert.

Tatsächlich haben wir den Text durch ein Hintergrundbild ersetzt, daher der (naheliegende) Begriff *Image Replacement*. Natürlich könnte das Hintergrundbild auch schicker als das von uns verwendete sein und verblichene Fotos, verschwommene Linien usw. einsetzen – alles, was Ihnen in den Sinn kommt.

Beachten Sie, dass es eine Reihe von Techniken zum Image Replacement gibt, bei denen viele den Text entfernen, indem das `span`-Element auf `display: none;` oder `visibility: hidden;` gesetzt wird. Bei anderen wird eine große negative Texteinrückung oder ein negativer linker Rand eingesetzt. Der von uns hier verwendete Positionierungsansatz ist eine weitere Technik.

Bilder vs. Image Replacement

Ist es sinnvoll, einen Hintergrund zu verwenden, um zu simulieren, was über ein Bild genau so gut erzielt werden kann? Schauen Sie sich das Markup an, das wir in diesem Projekt verwenden:

```
<h3><span>Eric's Thoughts</span></h3>
```

Vergleichen Sie das nun mit dem mehr prosaischen Ansatz plus das CSS, das wir geschrieben haben, damit das Bild erscheint:

```
<h3><img src="title.gif" alt="Eric's Thoughts"
  width="167" height="38"></h3>
```

Auf den ersten Blick mag das erstere effizienter erscheinen, aber ist es das auch wirklich? Nicht notwendigerweise. Betrachten Sie die Menge an Extra-CSS, die erforderlich war, um das `span`*-Element aus dem sichtbaren Bereich hinauszuschieben und das Bild in den Hintergrund von* `h3` *zu legen. Für einen einmaligen Effekt wie diesen ist die Verwendung eines* `img` *wahrscheinlich sinnvoller. Das ist dank des* `alt`*-Texts genauso zugänglich und »barrierefrei« und wirkt sich weniger belastend auf die Seitengröße insgesamt aus.*

Warum sollten wir uns also mit dem Ansatz des Image Replacements herumschlagen? Erst einmal ist er interessant und es wert, ausprobiert zu werden. Wichtiger noch ist aber, dass die Technik des Image Replacement sinnvoll ist, wenn sie auf mehreren Seiten verwendet wird. Nehmen Sie beispielsweise einmal an, dass der Firmenname auf jeder Seite stehen soll und Sie ihn durch ein Logo ersetzen wollen. Insgesamt werden Sie über die Herangehensweise mit dem Image Replacement mehr Bandbreite sparen als damit, auf jeder Seite die `img`*-Elemente zu einzubetten.*

Also ist Image Replacement für dieses spezielle Projekt, an dem wir gerade arbeiten, nicht die klügste Entscheidung, aber in vielen anderen Fällen kann sie sich als nützliche Technik herausstellen. Gerade deswegen sollte hier auch ihr Einsatz demonstriert werden.

8.4.2 Datum und Titel der Einträge

Jetzt, wo der Titel schön über allem thront, wird es Zeit, an den Einträgen zu arbeiten. Bei jedem Eintrag befindet sich oben ein Titel und das Datum, an dem der Text eingestellt wurde. Um beide einheitlich zu machen, wollen wir ihnen gemeinsame Styles geben: Ränder und Padding werden wir entfernen und beide auf Arial als Schrift erster Wahl einstellen (gefolgt von Verdana und dann jedem verfügbaren Sans-Serif-Font).

```
#weblog h3 span {position: absolute; left: -50em; width: 50em;}
#weblog h4.title, #weblog h5.date {margin: 0; padding: 0;
  font-family: Arial, Verdana, sans-serif; line-height: 1em;}
</style>
```

Wir können die Titel auf dreierlei Arten hervorheben. Am naheliegendsten ist, die Schriftgröße einfach ein wenig aufzustocken. Weniger naheliegend, aber genauso wichtig ist, oben bei den Titeln Padding einzufügen, um über dem Text etwas freien Raum zu schaffen. Wir entscheiden uns dafür, an der unteren Kante einen Rahmen zu ziehen.

```
#weblog h4.title, #weblog h5.date {margin: 0; padding: 0;
  font-family: Arial, Verdana, sans-serif; line-height: 1em;}
#weblog h4.title {font-size: 1.25em; padding: 5px 0 0;
  border-bottom: 1px solid rgb(50%,66%,50%);}
</style>
```

Momentan brauchen wir beim Datum nur die Schriftgröße etwas zu reduzieren und eine Farbe einzustellen – siehe Abbildung 8.5.

```
#weblog h4.title { font-size: 1.25em; padding: 5px 0 0;
  border-bottom: 1px solid rgb(50%,66%,50%);}
#weblog h5.date {font-size: 0.9em; color: rgb(50%,66%,50%);}
</style>
```

Zugegeben, das ist ein Fortschritt, aber bislang noch kein sonderlich schöner. Aber keine Bange – das wird gleich schon viel besser. Wir nehmen uns vor, Titel und Datum so wirken zu lassen, als ob sie an einem Blattstängel sitzen.

8.4.3 Aufteilen und neu zusammenführen

Was diese Einträge brauchen, ist ein richtig schöner Effekt für Titel und Datum. Da könnten wir beliebig viele Sachen versuchen, aber ganz interessant wäre es doch, wenn Titel und Datum sich sozusagen mittig treffen und ein Blatt jeweils an der Seite abschließt. Beispielsweise könnte der Titel links von einem Blatt sitzen und das Datum rechts davon.

Fragen zum Padding

Ein weiterer Grund, Padding statt Ränder einzusetzen, besteht darin, dass falls wir dem Titel je ein Hintergrundbild geben wollen, das Padding den Bereich erweitert, in dem es sichtbar wird. Warum wir Pixel verwenden, verraten wir gleich.

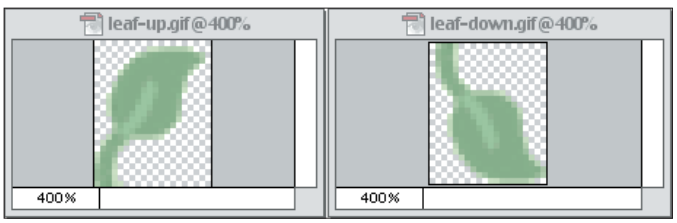
ABBILDUNG 8.5

Titel und Daten der Einträge nähern sich einander an und ändern ihr Erscheinen.

Dafür brauchen wir ein paar Bilder von Blättern. Wie wäre es mit denen aus der Abbildung 8.6? Ganz prima!

ABBILDUNG 8.6

Die Blattbilder, mit denen die Einträge herausgeputzt werden



Jetzt zum CSS. Zuerst nehmen wir uns den Titel vor. Wenn das nach oben gerichtete Blatt rechts neben dem Text sitzen wird, sollte der Text auch rechtsbündig sein.

```
#weblog h4.title {font-size: 1.25em; text-align: right;
  padding: 5px 0 0;
  border-bottom: 1px solid rgb(50%,66%,50%);}
```

Nun geben wir das Blatt dazu. Es soll rechts sein, also werden wir es im Hintergrund des Titels in der oberen rechten Ecke platzieren. Warum oben rechts statt unten rechts? Wenn das Bild höher als der Hintergrund ist, wird es auf diese Weise vom unteren Rahmen »abgeschnitten«, statt am oberen Titelrand zu verschwinden.

```
#weblog h4.title {font-size: 1.25em; text-align: right;
  padding: 5px 0 0;
  background: url(leaf-up.gif) 100% 0 no-repeat;
  border-bottom: 1px solid rgb(50%,66%,50%);}
```

Nach dieser Aktion ist es jedoch mehr als wahrscheinlich, dass der Titeltext das Hintergrundbild überlappen wird. Das wollen wir nicht, also werden wir dem Titel 25 Pixel Padding rechts zugeben. Und obendrein fügen wir noch einen großen linken Rand hinzu, damit der Titeltext in Richtung Layoutmitte geschoben wird. 45 % sollte etwa reichen; wir wollen nicht 50 % nehmen, weil sowohl der Titel als auch das Datum mittig im Layout sitzen soll. Den bisher erzielten Fortschritt sehen Sie in Abbildung 8.7.

```
#weblog h4.title {font-size: 1.25em; text-align: right;
  padding: 5px 25px 0 0; margin: 0 45% 0 0;
  background: url(leaf-up.gif) 100% 0 no-repeat;
  border-bottom: 1px solid rgb(50%,66%,50%);}
```

Okay, der blaue Link-Text fängt langsam an zu nerven, und den Unterstrich brauchen wir auch nicht wirklich. Das stellen wir beides ab.

```
#weblog h4.title {text-align: right; font-size: 1.25em;
padding: 5px 25px 0 0; margin: 0 45% 0 0;
  background: url(leaf-up.gif) 100% 0 no-repeat;
  border-bottom: 1px solid rgb(50%,66%,50%);}
#weblog h4.title a {text-decoration: none;
  color: rgb(15%,30%,15%);}
#weblog h5.date {font-size: 0.9em; color: rgb(50%,66%,50%);}
```

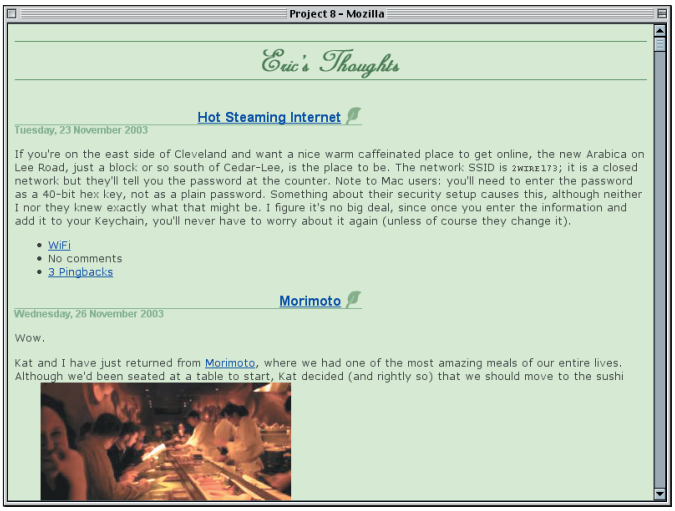


ABBILDUNG 8.7

Die Titel der Einträge erhalten ein Bild und Ränder.

Jetzt kümmern wir uns um die Datumseinträge. Im Prinzip müssen wir hier die Styles umkehren, die wir bei den Titeln angewendet haben. Das heißt also, ein linker Rand von 45 %, etwas Padding links, Text linksbündig und ein Hintergrundbild eines nach unten zeigenden Blattes, das in die linke untere Ecke im Hintergrund dieses Elements gestellt wird. Ach ja, ein oberer Rahmen fehlt auch noch. Das schaffen wir alles auf einmal. Bereit?

```
#weblog h5.date {font-size: 0.9em; text-align: left;
  padding: 0 0 5px 25px; margin: 0 0 0 45%;
  background: url(leaf-down.gif) 0 100% no-repeat;
  border-top: 1px solid; color: rgb(50%,66%,50%);}
```

Wir haben es im Grunde schon geschafft – aber da ist noch eine Kleinigkeit. Der untere Rahmen des Titels und der obere Rahmen des Datums scheinen sich in der Mitte aufeinander zu stapeln, was zu einer eigenartigen Kerbe führt, die nicht sehr schön aussieht. Es wäre besser, wenn die Rahmen tatsächlich zu einer einzigen Linie verschmelzen.

Wie kriegen wir das hin? Ganz einfach: Der Titel bekommt einen negativen unteren Rand von einem Pixel. Damit werden die Datumseinträge um einen Pixel nach oben verschoben und so überlappen die Rahmen. Optisch wird daraus eine einzelne Linie – siehe Abbildung 8.8.

```
#weblog h4.title {font-size: 1.25em; text-align: right;
  padding: 5px 25px 0 0; margin: 0 45% -1px 0;
  background: url(leaf-up.gif) 100% 0 no-repeat;
  border-bottom: 1px solid rgb(50%,66%,50%);}
```

Rundungsfehler

Aus irgendeinem Grund scheint die Gecko-Familie (Mozilla usw.) unter zufälligen Rundungsfehlern zu leiden, die die von uns hier geschaffene Ausrichtung aus der Bahn werfen kann. Wenn Padding und Ränder mit ems eingestellt werden, scheint sich das Problem zu verschlimmern, aber nicht bei der Verwendung von Pixeln. Darum haben wir sowohl für den Titel als auch das Datum ein pixelbasiertes Padding gewählt.

ABBILDUNG 8.8

Die Rahmen von Titel und Datum treffen dank eines negativen Randes aufeinander.

Um eine Sache müssen wir uns noch kümmern, bevor es weitergeht. Aus irgendeinem Grund wird der Internet Explorer für Windows die Berechnungen für die Ränder beim Titel und beim Datum durcheinander bringen, wenn wir alles so lassen. Dies mag mit dem berühmt-berüchtigten Bug des Internet Explorer für Windows zusammenhängen, bei dem die Ränder plötzlich verdoppelt werden, obwohl die Ränder gar nicht wirklich verdoppelt werden.

Auf jeden Fall kann dies dadurch entschärft werden, dass die Breite eines Container-Elements auf unter 100 % reduziert wird. Weil alle Einträge vom Weblog-`div` umschlossen werden, brauchen wir nur dessen Breite um einen Prozentpunkt verringern.

```
body {color: rgb(18%,19%,17%); background: rgb(85%,92%,81%);
 font: 0.85em Verdana, sans-serif;}
#weblog {width: 99%;}
#weblog h3 {font-size: 150%; text-align: center;
  color: rgb(10%,30%,10%);
  border: 1px solid rgb(30%,50%,30%); border-width: 1px 0;
  margin-bottom: 1.5em; height: 45px;
  background: url(title.gif) 50% 100% no-repeat;}
```

Die Lösung ist praktisch kaum erklärbar, aber das ist mit dem Bug ja genauso. Wie dem auch sei, wir können damit offensichtlich den Fehler im IE/Win lösen, also machen wir das auch so, schütteln mit dem Kopf und fahren fort.

8.4.4 Text und Information

Bei den Texten der Einträge brauchen wir eigentlich nicht sonderlich viel zu machen, aber den Einträgen selbst könnte es gut tun, wenn sie etwas auseinander gezogen werden. Weil jeder Eintrag von einem `div` eingeschlossen wird, können wir alles auseinander schieben, indem wir Ränder einfügen. Dazu nehmen wir einen unteren Rand.

```
#weblog h3 span {position: absolute; left: -50em; width: 50em;}
#weblog .entry {margin: 0 0 2em;}
#weblog h4.title, #weblog h5.date {margin: 0; padding: 0;
  font-family: Arial, Verdana, sans-serif; line-height: 1em;}
```

Allerdings können wir dem Texteintrag doch etwas Gutes tun. Wenn Sie den Text in Abbildung 8.8 genau untersuchen, scheint er irgendwie »zu locker geschüttet« zu sein, als wären die Buchstaben zu weit auseinander. Das Erscheinungsbild könnte besser wirken, wenn wir diesen Abstand etwas knapper halten. In der Typographie könnten wir ein automatisches Kerning einstellen oder gar an den Kerning-Werten selbst herumfeilen. CSS ist nicht so fortschrittlich, aber die Abstände zwischen den Zeichen können Sie doch darüber einstellen.

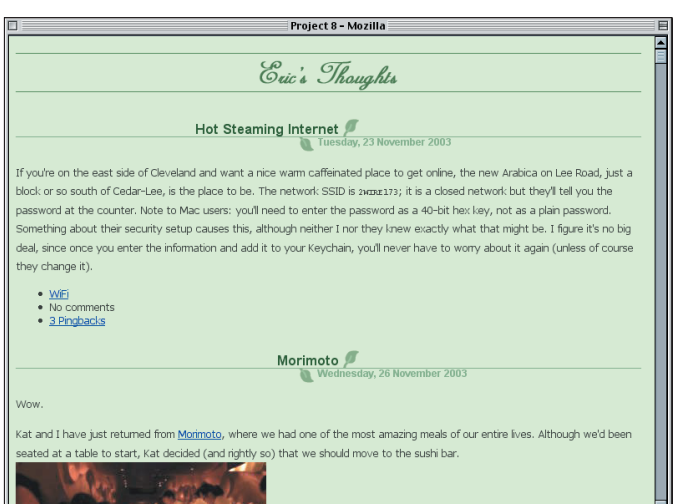
```
#weblog .entry {margin: 0 0 2em; letter-spacing: -1px;}
```

Doch mit dieser Regel ziehen wir auch den Text von Titel und Datum zusammen, und das wirkt gar nicht so gut. Also werden wir den Wert mit einer Regel überschreiben, die sowieso schon hier herumliegt.

```
#weblog h4.title, #weblog h5.date {margin: 0; padding: 0;
  font-family: Arial, Verdana, sans-serif; line-height: 1em;
  letter-spacing: 0;}
```

Noch eine Sache: Wir wollen den Zeilenabstand um eine Winzigkeit erhöhen, aber nur für Textabsätze. Andere Elemente wie Listen oder vorformatierten Text wird das nicht betreffen, aber das ist wahrscheinlich auch besser so. Das Ergebnis sehen wir in Abbildung 8.9.

```
#weblog h5.date {font-size: 0.9em; text-align: left;
  padding: 0 0 5px 25px; margin: 0 0 0 45%;
  background: url(leaf-down.gif) 0 100% no-repeat;
  border-top: 1px solid; color: rgb(50%,66%,50%);}
#weblog p {line-height: 1.4;}
</style>
```



Modifizierter Raum

Der Wert von `letter-spacing` ist ein Modifikator des Abstands zwischen Zeichen, nicht des reinen Abstands. Das hat damit zu tun, wie Zeichen-Glyphen in modernen Computersystemen gebildet werden, und das ist viel zu verzwickt, um es hier auszuführen. Denken Sie nur daran, dass `-1px` den Abstand (»Spacing«) um einen Pixel verringert, und `0` entspricht einem normalen, unmodifizierten Buchstabenabstand.

ABBILDUNG 8.9

Der Buchstabenabstand wird verringert und der Zeilenabstand erhöht.

Nun wenden wir uns der Information zu, die am Ende eines jeden Eintrags kommt. Wir können die »Sliding Doors«-Herangehensweise aus Projekt 7 »Hereinspaziert! Attraktive Tabs auf Ihrer Website« verwenden, damit das richtig klasse aussieht, aber das wäre eigentlich eher kontraproduktiv. Diese Information sollte sozusagen vielmehr in den Hintergrund treten und nicht mehr Aufmerksamkeit beanspruchen als der Haupteintrag.

Obwohl wir also nicht versuchen werden, etwas so Komplexes wie Tabs zu kreieren, wollen wir die drei Informationshappen in einer Linie erscheinen lassen, wobei alle visuell voneinander getrennt sind. Wir könnten sie beispielsweise durch vertikale Linien oder ähnliches voneinander trennen.

Wir beginnen damit, alle drei Listenelemente in einer Linie auszurichten.

```
#weblog h5.date {font-size: 0.9em; text-align: left;
  padding: 0 0 5px 25px; margin: 0 0 0 45%;
  background: url(leaf-down.gif) 0 100% no-repeat;
  border-top: 1px solid; color: rgb(50%,66%,50%);}
#weblog .moreinfo li {display: inline;}
#weblog p {line-height: 1.4;}
```

Dies wird die Listenelemente dazu bringen, Inline-Boxen zu generieren, genau wie die in ihnen enthaltenen Links es tun. Das sollte an sich schon ausreichen, um die Marker der Listenelemente (die Bullets) zum Verschwinden zu bringen, aber das macht nicht jeder Browser mit, also kümmern wir uns darum, dass sie verschwinden.

```
#weblog .moreinfo li {display: inline; list-style: none;}
```

Nun nehmen wir uns die visuelle Trennung vor. In vielen Fällen werden Autoren den vertikalen Strich (|) nutzen, um eine Reihe nebeneinander angeordneter Links zu trennen. Uns stehen keine Zeichen zur Verfügung, aber wir können doch die Kanten der Inline-Boxen der Listenelemente nehmen. Wenn wir also jedem Listenelement einen linken Rahmen geben und noch etwas Padding links dazu, dann kriegen wir das in Abbildung 8.10 gezeigte Ergebnis.

```
#weblog .moreinfo li {display: inline; list-style: none;
  border-left: 1px solid rgb(65%,75%,65%);
  padding-left: 0.5em;}
```

Wenn wir uns diese Änderung genauer anschauen, befinden sich die Rahmen zu dicht am benachbarten Text. Der Raum beispielsweise zwischen »WiFi« und dem rechten Rahmen (das ist der linke Rahmen des Listenelements »No comments«) ist zu schmal. Also wollen wir die letzte Deklaration derart ändern, dass wir bei diesem Listenelement sowohl links als auch rechts ein halbes em Padding einfügen – aber nicht oben und unten.

```
#weblog .moreinfo li {display: inline; list-style: none;
  border-left: 1px solid rgb(65%,75%,65%);
  padding: 0 0.5em;}
```

Spiel mit Listenpunkten

Sie könnten dazu verleitet sein, die Listenpunkte zum Bleiben zu zwingen, weil sie sich gut als Separatoren zwischen den Listenelementen eignen. Das Problem dabei ist, dass Inline-Boxen keine Listenpunkte generieren, auch wenn man das explizit angibt. Listenpunkte erscheinen nur bei Listenelement-Boxen, zumindest in CSS-konformen Browsern.

Abstände

Im Internet Explorer 5 für Windows könnten die Abstände ein wenig verschoben sein. Das liegt an dessen Umgang mit dem Whitespace zwischen den Listenelementen und kann durch ein Abändern des gerade eingefügten Paddings reduziert werden.

ABBILDUNG 8.10

Die Listenelementen werden in einer Zeile Text aufgereiht.

Wichtiger noch – wenigstens im Moment – ist die Einrückung der Listenelemente. Das liegt am `ul`-Element, in denen sie enthalten sind, das immer noch einen linken Rand hat (oder in Gecko-basierten Browsern: ein Padding links). Das entfernen wir nun vollständig.

```
#weblog h5.date {font-size: 0.9em; text-align: left;
  padding: 0 0 5px 25px; margin: 0 0 0 45%;
  background: url(leaf-down.gif) 0 100% no-repeat;
  border-top: 1px solid; color: rgb(50%,66%,50%);}
#weblog .moreinfo {margin: 0; padding: 0;}
#weblog .moreinfo li {display: inline; list-style: none;
  border-left: 1px solid rgb(65%,75%,65%);
  padding: 0 0.5em;}
```

Jetzt die ganz große Frage: Wollen wir, dass die Links auf der linken Seite stehen? Das ist sicher eine vertretbare Designentscheidung, aber lassen Sie uns die Links lieber nach rechts verschieben. Das hebt sie deutlich vom Haupttext ab und versetzt uns die Lage, dieser Gruppe von Links noch das letzte I-Tüpfelchen zu verpassen. Weil die Listenelemente nun Inline-Boxen generieren, wirken sich wie bei jedem andern Inline-Text Werte bei `text-align` auf sie aus.

```
#weblog .moreinfo {margin: 0; padding: 0;
  text-align: right;}
```

Jetzt zur Krönung noch die Links! Erinnern Sie sich daran, dass wir allen Listenelementen einen linken Rahmen gegeben haben? Weil sie alle nach rechts gerichtet sind, können wir die rechte Kante des `ul`-Elements nutzen, um sozusagen einen nachfolgenden Separator einzufügen. Dies wird in Abbildung 8.11 gezeigt.

```
#weblog .moreinfo {margin: 0; padding: 0;
text-align: right;
  border-right: 1px solid rgb(50%,66%,50%);}
```

ABBILDUNG 8.11

Die Links werden nach rechts verschoben und mit einem Rahmen »gekrönt«.

Okay, das ist gut, aber die Farben sind murks. Das Blau muss wirklich endlich weg. Nicht schwer, daraus einen Grünton zu machen.

```
#weblog .moreinfo li {display: inline; list-style: none;
  border-left: 1px solid rgb(65%,75%,65%);
  padding: 0 0.5em;}
#weblog .moreinfo a {color: rgb(50%,66%,50%);}
#weblog p {line-height: 1.4;}
```

Rahmen und Farbe

Da wir nun den gleichen `color`-Wert wie beim Rahmen hinzugefügt haben, können wir den Farbwert aus der `border-right`-Deklaration entfernen. Denn wenn der Rahmen eines Elements keine explizit definierte Farbe aufweist, wird der Wert aus dem Wert für `color` des gleichen Elements entnommen. Wir werden das so belassen, aber es ist ein Verhalten, das man im Hinterkopf behalten sollte.

Klasse, das Blau ist weg, aber der Text »No comments«, der nicht in einem Link steht, ist immer noch schwarz. Zum Glück können wir das auch ganz leicht abändern. Wir brauchen bloß die gewünschte Farbe entweder den Listenelementen oder dem `ul`-Element selbst zuweisen. Weil `color` vererbt wird, wird der von uns deklarierte Wert auf den Nicht-Link-Text angewendet. Da den Links explizit eine Farbe zugewiesen wurde, werden sie sich nicht ändern. Das ist mehr oder weniger egal, aber wir werden die Farbe dem `ul`-Element zuweisen.

```
#weblog .moreinfo {margin: 0; padding: 0;
  text-align: right;
  border-right: 1px solid rgb(50%,66%,50%);
  color: rgb(50%,66%,50%);}
```

Nun wollen wir über eine ähnliche Technik jeden Text, der kein Link ist, auf kursiv stellen, aber bei allen Links den normalem Font-Style lassen. Wir können eine Deklaration zum Kursivstellen des gesamten Texts in der Liste wie folgt einfügen:

```
#weblog .moreinfo {margin: 0; padding: 0;
  text-align: right; font-style: italic;
  border-right: 1px solid rgb(50%,66%,50%);
  color: rgb(50%,66%,50%);}
```

Augenblicklich werden Links so wie Nicht-Link-Text auf kursiv gesetzt. Um ihr normales Aussehen wieder herzustellen, stellen wir für die Links `font-style` explizit auf `normal`. (Denken Sie daran, dass die Separatoren Rahmen sind, also wird sich die Änderung auf Kursiv bei ihnen nicht auswirken.)

```
#weblog .moreinfo a {color: rgb(50%,66%,50%); font-style: normal;}
```

Bevor wir weitermachen, erledigen wir noch eine weitere Sache: Fettschrift für den Link für die Kategorie (»WiFi«). Das geht ganz einfach: eine `font-weight`-Deklaration auf das Listenelement mit dem Klassennamen `categories` anwenden – siehe Abbildung 8.12.

```
#weblog .moreinfo a {color: rgb(50%,66%,50%); font-style: normal;}
#weblog .moreinfo .categories {font-weight: bold;}
#weblog p {line-height: 1.4;}
```

Abbildung 8.12

Gemischter Text: kursiv, fett und normal

Der Link »No comments«

In einem echten Weblog wäre der Text »No comments« wahrscheinlich der Link zu einem Kommentar-Formular. Wir belassen ihn hier so als Nicht-Link, um das Styling der Liste und der Links zu illustrieren und den Text ohne Link anders ausschauen zu lassen.

8.4.5 Der letzte Feinschliff

Nun ist es bald geschafft, aber da ist noch eine größere Geschichte, um die wir uns zu kümmern haben. Schauen Sie sich das Bild im zweiten Eintrag an: Es sitzt mitten in einem Textabsatz und verschandelt uns das Layout. Das muss behoben werden.

Wir könnten nun alle in einem Eintrag vorkommenden Bilder in Floats verwandeln, aber es könnte sein, dass ein Bild eben gerade nicht gefloatet werden soll (beispielsweise ein kleines Icon). Wir brauchen also dafür noch einen Hook, um das Bild zu floaten, und zum Glück haben wir da noch einen. Achten Sie auf das Attribut `class` genau vor der abschließenden Klammer.

```
<img src="morimoto.jpg" alt="A view of the sushi bar from our seats,
  with Morimoto and his sushi staff slicing away"
  title="Morimoto's Sushi Bar" class="pic border">
```

Multiclassing

Das Attribut `class` akzeptiert nicht bloß zwei, sondern beliebig viele durch Leerzeichen getrennte Klassennamen. Alle davon können in einem Selektor verwendet werden. Theoretisch können Sie Elemente danach auswählen, ob sie zwei oder mehr Klassennamen haben (der Selektor `.pic.border` wählt beispielsweise alle Elemente mit einer `class`, in der sowohl `pic` als auch `border` enthalten ist), aber der IE/Win unterstützt diese Fähigkeit nicht. Er kann jedoch nur mit einem Wort zur Zeit umgehen, so wie wir das hier auch gemacht haben.

Also hat das Bild einen `class`-Wert, in dem zwei Namen enthalten sind: `pic` und `border`. Technisch gesprochen sind dies keine separaten `class`-Werte, aber separate Namen, und wir können sie individuell ansprechen. Alle `pic`-Bilder sollen als Float nach rechts gesetzt werden und Ränder bekommen, damit der Text nicht so dicht um das gefloatete Bild herumfließt.

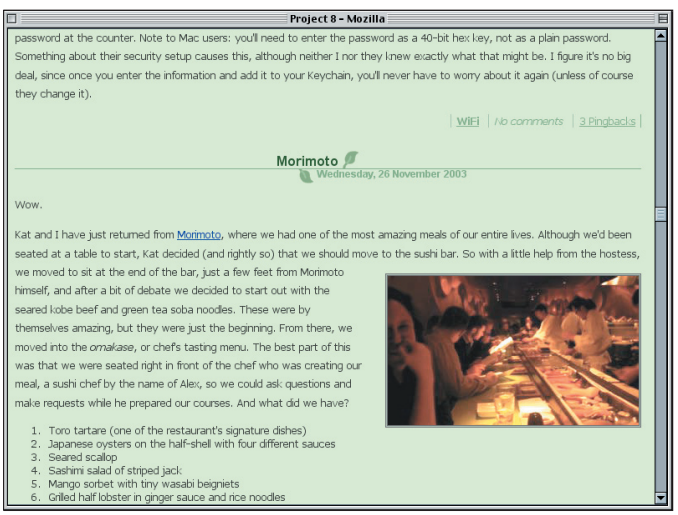
```
#weblog p {line-height: 1.4em;}
#weblog .pic {float: right; margin: 1em 0 1em 2em;}
</style>
```

Der `class`-Name `border` weist darauf hin, dass bei dem Bild ein Rahmen eingefügt werden soll. Wir machen es nun besonders schick und geben dem Bild einen grauen doppelten Rahmen und auch einen hellgrünen Hintergrund. Diesen Effekt können Sie in Abbildung 8.13 bewundern.

```
#weblog .pic {float: right; margin: 1em 0 1em 2em;}
#weblog .border {border: 3px double #666; background: #ABA;}
</style>
```

ABBILDUNG 8.13

Das Bild mit Rahmen als Float



Genau, Bilder können Hintergründe haben! Und da sich obendrein der Hintergrund bis zur äußeren Randkante des Elements erstreckt, wird er in der Lücke zwischen den beiden Linien des doppelten Rahmens sichtbar. Subtil, aber raffiniert.

Um das Projekt nun langsam abzuschließen, wollen wir noch etwas ganz Abgefahrenes hinzufügen. Die meisten Browser werden das nicht erkennen, aber denen fehlt dabei dann auch nichts.

Nehmen wir uns noch einmal die Titel der Einträge vor, insbesondere die Links darin. Diese Links sind sogenannte »Permalinks«, also Verweise auf archivierte Kopien des Eintrags. Darüber kann auf diesen Eintrag verlinkt werden, ohne befürchten zu müssen, dass er irgendwann mal verschwunden ist. Ein Permalink wird üblicherweise mit einem Doppelkreuz (#) gekennzeichnet. Es wäre doch interessant, wenn eine Raute erscheint, falls der User mit der Maus über den Permalink fährt, oder? Wir lassen sie vor dem Link erscheinen und geben ihr auch eine hellere Farbe als dem Link-Text.

```
#weblog h4.title a {text-decoration: none;
  color: rgb(15%,30%,15%);}
#weblog h4.title a:hover:before {content: "# ";
  color: rgb(50%,61%,48%);}
#weblog h5.date {font-size: 0.9em; text-align: left;
  padding: 0 0 5px 25px; margin: 0 0 0 45%;
  background: url(leaf-down.gif) 0 100% no-repeat;
  border-top: 1px solid; color: rgb(50%,66%,50%);}
```

Damit wird im Prinzip der String, den wir deklariert haben (ein Doppelkreuz, gefolgt von einem Leerzeichen), am Anfang des Links als Inhalt eingefügt. Auf diese Weise eingefügte Inhalte werden als *generierter Inhalt* bezeichnet. Der generierte Inhalt wird auf den deklarierten grünen Farbton eingestellt, ohne die Farbe des Links insgesamt zu ändern.

Zum Zeitpunkt dieses Manuskripts werden Sie den Effekt nur in den jüngsten Versionen von Mozilla (1.5+) und Opera (7+) sehen können. Safari 1.1+ sollte ihn auch zeigen, aber damit das funktioniert, brauchen wir einen kleinen Hack. Wenn wir dem Link selbst einen Rahmen zugeben, wird der generierte Inhalt plötzlich erscheinen.

```
#weblog h4.title a {text-decoration: none;
  color: rgb(15%,30%,15%);}
#weblog h4.title a:hover {border-top: 1px dotted rgb(50%,61%,48%);}
#weblog h4.title a:hover:before {content: "# ";
  color: rgb(50%,61%,48%);}
```

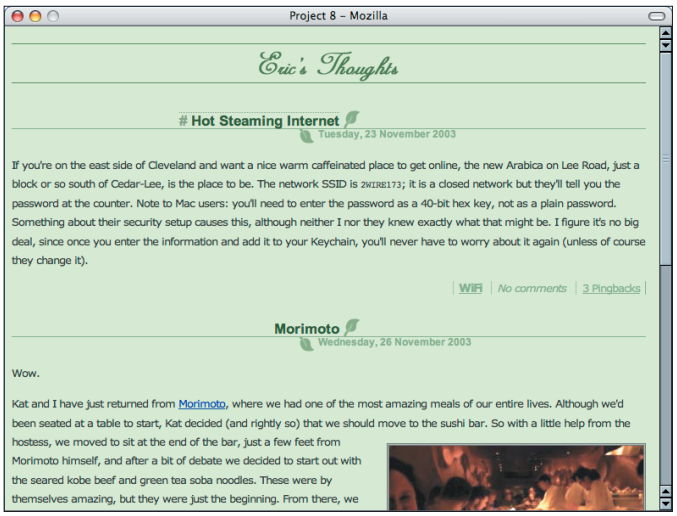
Es ist nicht klar, warum Safari über das Einsetzen eines Rahmens beim Link dazu gebracht werden kann, das Gewünschte zu machen, aber es klappt. Als Bonus wird der Rahmen in den meisten Browsern erscheinen – auch im Internet Explorer, der aber den generierten Inhalt nicht anzeigt. Somit werden Sie in fortschrittlicheren Browsern das Ergebnis aus Abbildung 8.14 sehen, und das verdanken wir alles dem Stylesheet, das in Listing 8.2 aufgeführt ist.

Heia Safari

Die Unterstützung für generierten Inhalt in Safari 1.2 ist verbessert worden; darum ist es nicht mehr länger nötig, einen Rahmen einzufügen, damit der generierte Inhalt erscheint. Der Rahmen wird nur noch bei Safari 1.1 gebraucht.

Abbildung 8.14

Die Titel-Links erhalten dynamische Styles.



Listing 8.2 Das vollständige Stylesheet

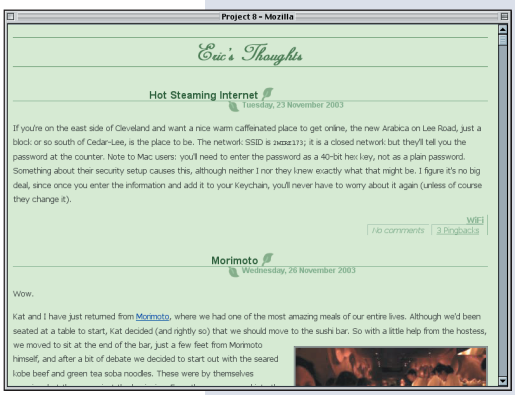
```
body {color: rgb(18%,19%,17%); background: rgb(85%,92%,81%);
 font: 0.85em Verdana, sans-serif;}
#weblog {width: 99%;}
#weblog h3 {font-size: 150%; text-align: center;
  color: rgb(10%,30%,10%);
  border: 1px solid rgb(30%,50%,30%); border-width: 1px 0;
  margin-bottom: 1.5em; height:45px;
  background: url(title.gif) 50% 100% no-repeat;}
#weblog h3 span {position: absolute; left: -50em; width: 50em;}
#weblog .entry {margin: 0 0 2em; letter-spacing: -1px;}
#weblog h4.title, #weblog h5.date {margin: 0; padding: 0;
  font-family: Arial, Verdana, sans-serif; line-height: 1em;
  letter-spacing: 0;}
#weblog h4.title {font-size: 1.25em; text-align: right;
  padding: 5px 25px 0 0; margin: 0 45% -1px 0;
  background: url(leaf-up.gif) 100% 0 no-repeat;
  border-bottom: 1px solid rgb(50%,66%,50%);}
#weblog h4.title a {text-decoration: none;
  color: rgb(15%,30%,15%);}
#weblog h4.title a:hover {border-top: 1px dotted rgb(50%,61%,48%);}
#weblog h4.title a:hover:before {content: "# ";
  color: rgb(50%,61%,48%);}
#weblog h5.date {font-size: 0.9em; text-align: left;
  padding: 0 0 5px 25px; margin: 0 0 0 45%;
  background: url(leaf-down.gif) 0 100% no-repeat;
  border-top: 1px solid; color: rgb(50%,66%,50%);}
#weblog .moreinfo {margin: 0; padding: 0;
  text-align: right; font-style: italic;
  border-right: 1px solid rgb(50%,66%,50%);
  color: rgb(50%,66%,50%);}
#weblog .moreinfo li {display: inline; list-style: none;
```

```
  border-left: 1px solid rgb(65%,75%,65%);
  padding: 0 0.5em;}
#weblog .moreinfo a {color: rgb(50%,66%,50%); font-style: normal;}
#weblog .moreinfo .categories {font-weight: bold;}
#weblog p {line-height: 1.4;}
#weblog .pic {float: right; margin: 1em 0 1em 2em;}
#weblog .border {border: 3px double #666; background: #ABA;}
```

8.5 Spielwiese

Hier folgen nun einige stilistische Variationen, die Sie einmal ausprobieren können:

1. Ändern Sie die Zusatzinformation so ab, dass die Kategorie auf einer eigenen Zeile über den beiden anderen Infohäppchen sitzt. Versuchen Sie, auch bei den Elementen »comment« und »pingback« einen Rahmen einzufügen, und achten Sie darauf, dass der Rahmen von »pingback« nicht den Rahmen auf seiner rechten Seite berührt.



2. Werfen Sie die Ränder bei den Eintragstiteln hinaus und verwenden Sie stattdessen Padding und Hintergrund-Positionierung, um den Text des Titels und das Hintergrundbild so zu platzieren, wie es im Projekt ausgesehen hat. Dazu wird es auch nötig sein, einen weiteren Rahmen zu entfernen.

3. Hier kommt eine doppelte Änderung. Dunkeln Sie zuerst den Titel des Weblogs über einige Änderungen des Hintergrunds und des Rahmens ab. Verschieben Sie danach die Rahmen des Bildes um ein Pixel nach außen, damit der Hintergrund zwischen dem Bild und dem doppelten Rahmen sichtbar wird.

9

Design einer Homepage

Gutes Design gibt dem User Zufriedenheit, dem Hersteller schwarze Zahlen und dem Ästheten keinen Grund zum Anstoß.

Raymond Loewy

Das Styling eines Weblogs wie im vorigen Projekt stellt eine Herausforderung dar, aber nur ein Weblog-Style genügt in der Regel nicht. Weblog-Einträge erscheinen wahrscheinlich im Kontext eines Website-Designs, und beides muss Hand in Hand gehen. Schließlich sähe es recht merkwürdig aus, wenn sich im Weblog ein blättriges Naturthema abbildet und der Rest des Designs für die Website bestünde aus einer Vielzahl von neonfarbenen J-Pop-Dekorationen. Noch befremdlicher wäre es, wenn beides auf der gleichen Seite zusammenträfe.

Da wir bereits über das Design für das Weblog verfügen, können wir einfach weiterarbeiten und ein Website-Design kreieren, das zu den Weblog-Styles passt, anstatt es anders herum zu versuchen. Somit können wir auf dem bisher Geschaffenen aufbauen und uns beispielhaft erarbeiten, wie die verschiedenen Komponenten einer Seite zusammenpassen können.

9.1 Projektziele

Aus der Natur (kein Wortspiel beabsichtigt!) dieses Projekts lassen sich direkt keine speziellen Ziele ableiten. Tatsächlich können wir die Gesamtheit der notwendigen Aufgaben in drei Punkten zusammenfassen:

- Das Weblog, das wir in Projekt 8 gestylt haben, wollen wir in den Zusammenhang einer ganzen Homepage stellen. Das bedeutet, den Masthead (Seitenkopf), die Seitenleiste und die Fußzeile so zu stylen, dass sie zum visuellen Thema passen, für das wir uns bei der Erarbeitung des Weblogs entschieden haben.
- Die Links für die Navigation und Präsentation sollen rechts vom Hauptinhalt platziert werden, und diese »Seitenleiste« soll dann oben am Masthead beginnen.
- Der Name des aktuellen Themas für die Website soll hervorgehoben werden.

Dies sind zugegebenermaßen recht allgemeine Richtlinien, aber sie reichen aus, um uns zu einem abschließenden Design zu führen.

9.2 Vorbereitungen

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

Laden Sie die Dateien für Projekt 9 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch09proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und erneut laden.

9.3 Die Grundlagen

Importbeschränkungen

Wie Sie sehen werden, erscheinen alle Styles, die wir in diesem Projekt hinzufügen, nach dem `@import`. Das ist kein Zufall. Die CSS-Spezifikation erfordert, dass alle `@import`-Angaben am Anfang des Stylesheets zu erscheinen haben – vor allen anderen Regeln!

Da wir die Styles aus dem Weblog-Projekt übernehmen werden, haben wir bereits ein einfaches eingebettetes Stylesheet, das in Listing 9.1 als Teil der allgemeinen Dokumentstruktur gezeigt wird.

Listing 9.1 Die grundlegende Dokumentstruktur und das Ausgangs-Stylesheet

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Project 9</title>
 <style type="text/css">
  @import url(project08.css);
 </style>
</head>
```

```
<body>
 <div id="sitemast">
 <h1>
  <a href="http://www.meyerweb.com/"><span>meyerweb</span>.com</a>
 </h1>
 </div>
 <div id="main">
  <div class="skipper">
   Skip to: <a href="#navpres">site navigation/presentation</a>
  </div>
  <div id="weblog">
   [...Weblog-Inhalte...]
  </div>
 </div>
 <div class="panel" id="navpres">
  [...Navigations- und Präsentationslinks...]
 </div>
 <div id="footer">
  [...Fußzeileninhalt...]
 </div>
</body>
</html>
```

Das eingebettete Stylesheet verwendet ein `@import`, um das Stylesheet aus Projekt 8 einzubinden, das wir in eine separate Datei namens `project08.css` gelegt haben. (Sie können sich dieses Stylesheet im Listing 8.2 des vorigen Projekts anschauen.) Das Ergebnis sehen wir in Abbildung 9.1.

ABBILDUNG 9.1

Dieses Design verfügt bisher nur über die Weblog-Styles.

Bisher haben wir also bloß den Weblog-Inhalt aus dem vorigen Projekt genommen und ihn in ein etwas größeres Dokument gelegt, eines mit Masthead und Navigationselementen plus Präsentationsoptionen und einem Footer (Fußzeile).

Die Seite hat auch einen »Skip-Link« ganz oben im Dokument; das ist der Teil im `div`-Element mit dem Individualformat `skipper`. Durch ein Anklicken (oder eine andere Aktivierung) lässt der Skip-Link die Browseranzeige zu den Navigations- und Präsentations-Links springen, die sich am Ende des Dokuments finden. Dies wird in Abbildung 9.2 gezeigt.

ABBILDUNG 9.2

Der Zielort des Skip-Links

Die Position dieser Links im Quelldokument macht eindeutig klar, dass wir entweder den Hauptinhalt nach links floaten müssen oder die »Seitenleiste« absolut zu positionieren haben, damit sie auf der rechten Seite des Hauptinhalts bleibt, wie es unsere Projektziele erfordern.

Ich setze die »Seitenleiste« in Anführungszeichen, weil nichts in der Dokumentstruktur darauf hinweist, dass sie zu einer Seitenleiste gemacht werden muss. Wir können sie beliebig stylen – als eine Reihe von Drop-down-Menüs vielleicht, wie wir das in Projekt 6 »Drop-down-Menüs mit CSS« gemacht haben. Es könnte auch alles ganz am Ende der Seite bleiben. Wir sprechen von den Navigations- und Präsentations-Links hauptsächlich deswegen als Seitenleiste, weil wir so bequemer auf unsere Absicht verweisen können, diese Links seitlich neben den Hauptinhalt zu legen.

9.4 Das Design wird erstellt

Wie gewöhnlich werden wir auch hier ganz oben im Dokument beginnen und uns nach unten durcharbeiten. Nachdem wir den Masthead erstellt haben, machen wir beim Hauptinhalt und der Seitenleiste weiter. Damit wird das Layout effektiv in zwei Bereiche geteilt: »Masthead« und »der Rest der Seite«. Zwar werden Hauptinhalt und Seitenleiste in Relation zueinander platziert, aber beide werden auch in Relation zum Masthead platziert, also ist dieser in vielerlei Hinsicht der wichtigste Aspekt des Designs.

9.4.1 Zwei Bilder hinterm Masthead

Weil der Masthead im Design eine Schlüsselfunktion einnimmt, versteht es sich von selbst, dass er gut aussehen muss. Es ist schwer, ohne Bilder ein schönes Webdesign zu schaffen, und darum gestatten wir uns zwei Bilder für den Masthead. Sie werden in Abbildung 9.3 gezeigt.

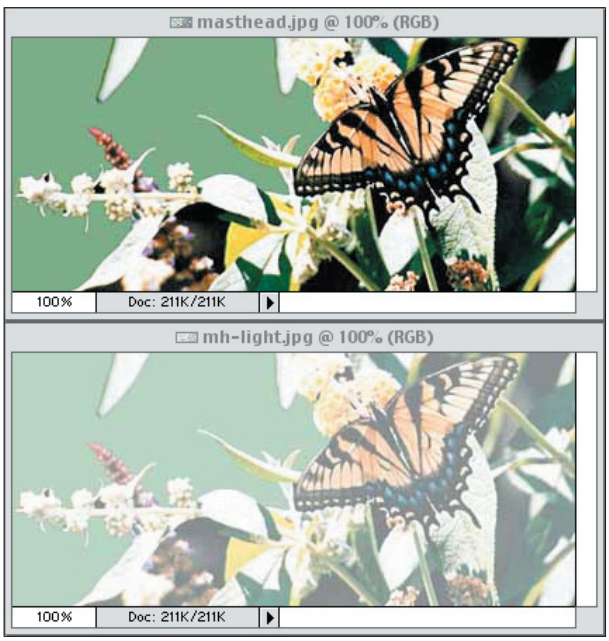


Abbildung 9.3
Die verfügbaren Bilder für den Masthead

Zugegeben, es sind eigentlich zwei Varianten des gleichen Bildes, `masthead.jpg` und `mh-light.jpg`, wobei es sich beim letzteren um eine aufgehellte Version des ersteren handelt. Wir werden beide nutzen, um einen Transparenzeffekt zu kreieren, wie wir ihn aus dem Projekt 4 »Positionieren im Hintergrund« kennen.

Zuerst wollen wir das `h1`-Element stylen. Es bekommt `masthead.jpg` als Hintergrund, also sollten wir seine Hintergrundfarbe an den Hintergrund des Bildes anpassen. Wir werden das Bild dann an die rechte Kante des Elements stellen.

```
@import url(project08.css);
#sitemast h1 {background: rgb(45%,65%,45%) url(masthead.jpg)
   100% 0 no-repeat;}
</style>
```

Aus Gründen, die später deutlicher werden (und im Projekt 4 erörtert wurden), müssen wir beim `h1`-Element die `font-size` explizit einstellen und alle Ränder oder Paddings definieren. Diese werden alle über ems eingestellt – aus Gründen der Konsistenz und auch, damit wir es später leichter haben. Zuerst einmal also das Padding und der Rand:

```
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% 0 no-repeat;}
```

Und nun zur Größe des `h1`-Fonts. Es erscheint als passend, ihn doppelt so groß zu setzen wie sein Elternelement (das sitemast-`div`). Das ist groß genug, um leicht erkennbar zu sein, aber nicht so groß, dass alles andere erschlagen wird. Damit alles einheitlich bleibt, soll die Schriftfamilie auch entsprechend der Titel aus unserem Weblog eingestellt werden. Das führt uns zu den folgenden Ergänzungen:

```
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: 2em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% 0 no-repeat;}
```

Das wird genau das Ergebnis bringen, das wir deklariert haben, *und nicht mehr* – das bedeutet, der Text wird nicht in Fettschrift angezeigt. Unsere neue Deklaration stellt `font-weight`, `font-style` und `font-variant` alle auf `normal`. Mit dieser Notation können verschiedene Schriftformatierungen in einer Regel zusammengefaßt werden. Also geben wir der Deklaration noch das `bold` hinzu und stellen die Fettschrift wieder her – siehe Abbildung 9.4.

```
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: bold 2em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% 0 no-repeat;}
```

ABBILDUNG 9.4

Das Erscheinungsbild des Mastheads verbessert sich dramatisch.

Das sieht schon alles ziemlich gut aus, aber ein paar Verbesserungen sind noch möglich. Zuerst lassen wir die Unterlänge des »y« aus dem Hintergrund des `h1`-Elements hervorragen. Das können wir erreichen, indem wir die `line-height` des Elements auf einen ausreichend niedrigen Wert stellen. Nach ein wenig Experimentieren finden wir heraus, dass `0.75em` für Arial eine gute Wahl ist. Wir machen uns die einzigartige Fähigkeit von `font` zunutze, nach dem Wert für `font-size` einen Wert für `line-height` zu akzeptieren, wenn beide durch einen Slash (/)getrennt sind.

```
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: bold 2em/0.75em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% 0 no-repeat;}
```

Jetzt machen wir uns an den Link selbst. Es mutet irgendwie komisch an, normale Link-Styles im Masthead vorzufinden, also werden wir die Farbe verändern und den Unterstrich herausnehmen.

```
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: bold 2em/0.75em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% 0 no-repeat;}
#sitemast h1 a {color: rgb(20%,40%,20%); text-decoration: none;}
</style>
```

Nun zum Masthead-`div` selbst. Weil das Bild im `h1` die normalen Farben hat, werden wir die verblasste Version im `div`-Element anwenden.

```
@import url(project08.css);
#sitemast {margin: 0;
  background: rgb(73%,82%,73%) url(mh-light.jpg) 100% 0 no-repeat;}
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: bold 2em/0.75em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% 0 no-repeat;}
```

Alles klar, nun haben wir also ein `div`-Element, das ein `h1` umschließt. Das bedeutet, dass vom `div`-Element momentan wirklich nichts zu sehen ist; hätten wir Rahmen eingefügt, wären sie sichtbar gewesen, aber das hilft dem Hintergrund auch nicht viel. Wir müssen also etwas vom Hintergrund sichtbar werden lassen. Kinderspiel: Wir werden dem `div`-Element Padding unten geben, und wie in Abbildung 9.5 zu sehen ist, wird vom Hintergrund nun etwas sichtbar.

```
#sitemast {margin: 0; padding: 0 0 1em;
  background: rgb(73%,82%,73%) url(mh-light.jpg) 100% 0 no-repeat;}
```

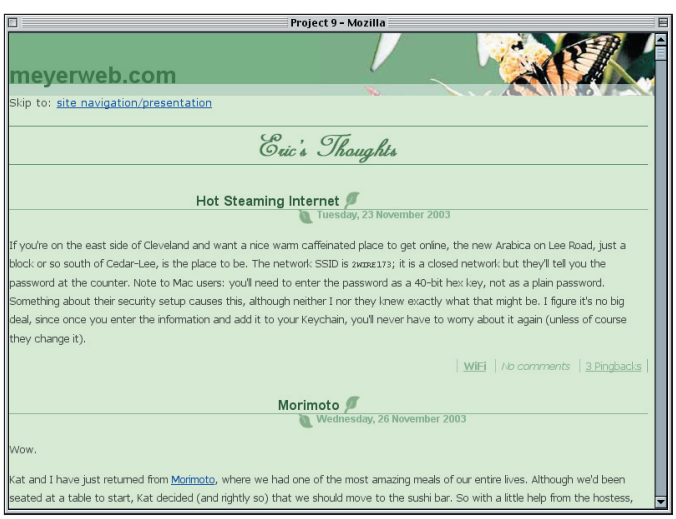


ABBILDUNG 9.5

Das hellere Bild wird in das Masthead-`div` eingefügt und sichtbar gemacht.

Verschieben vs. Bildbearbeitung

Wenn wir beschließen, dass das Erscheinungsbild des Mastheads so wie gewünscht ist, sollten wir die Bilder mit einem Bildbearbeitungsprogramm wie Photoshop beschneiden, damit ihre oberen Kanten so sind, wie wir sie hier im Design sehen. Wir können dann die Hintergrundpositionen auf `100% 0` setzen und noch dazu etwas Bandbreite sparen. Bis dahin können wir viel Zeit sparen, wenn wir verschiedene Bildanordnungen über `background-position` ausprobieren.

9.4.2 Rahmen und Verschiebungen

Allem Anschein nach ist der Masthead nun schon verflixt gut in Form, also können wir den nächsten Schritt angehen. Wir wollen mehr von dem Schmetterling zeigen, und zwar, indem wir ihn nach oben verschieben. Das können wir machen, indem wir die Positionen der Hintergrundbilder sowohl im `h1`- als auch im `div`-Element ändern.

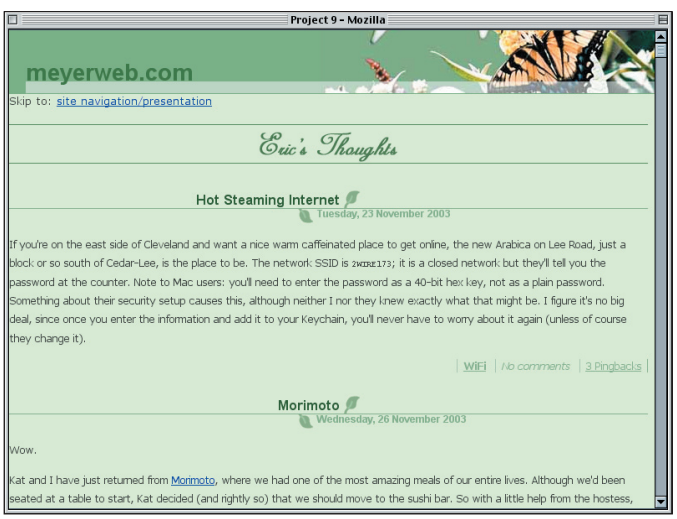
```
#sitemast {margin: 0; padding: 0 0 1em;
  background: rgb(73%,82%,73%) url(mh-light.jpg) 100% -30px no-repeat;}
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: bold 2em/0.75em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% -30px no-repeat;}
```

Einen letzten Pinselstrich sollte der Masthead aber noch kriegen. Indem wir die Rahmen kreativ einsetzen, können wir das `h1`-Element und den sichtbaren Bereich des `div`-Elements tatsächlich verschieben. Überlegen Sie sich mal die hier gezeigten Styles und deren Wirkung in Abbildung 9.6.

```
#sitemast {margin: 0; padding: 0 0 1em;
  border: 1px solid rgb(45%,65%,45%); border-width: 0 0 1px 1.5em;
  background: rgb(73%,82%,73%) url(mh-light.jpg) 100% -30px no-repeat;}
```

ABBILDUNG 9.6

Letzte Feinarbeit am Masthead



Das funktioniert, weil wir dem Masthead-`div` einen linken Rahmen von 1,5 em gegeben und die Farbe dieses Rahmens auch passend zur Hintergrundfarbe des `h1`-Elements gemacht haben. Visuell gehen die beiden praktisch nahtlos ineinander über. Wenn der Rahmen irgendeine andere Farbe erhält, wird sofort deutlich, wie dieser Effekt funktioniert.

Der 1-Pixel-Rahmen unten am Masthead dient auch dazu, ihn ganz dezent vom Rest des Designs abzuheben. Da wir gerade davon sprechen – es wird Zeit, dass wir uns dem Rest des Designs zuwenden.

9.4.3 Inhalt und Seitenleiste

Wie Sie sich sicher erinnern, steht der Hauptinhalt der Seite in einem `div`-Element mit dem Individualformat (`id`) `content`, und die Seitenleiste steht in einem `div`-Element mit dem Individualformat `navpres` und hat die Klasse `panel`. Unsere Projektziele weisen uns nun aber auch darauf hin, dass die Navigations- und Präsentations-Links rechts vom Hauptinhalt zu stehen haben.

Damit das geschehen kann, müssen wir eine der beiden folgenden Sachen tun:

- Den Hauptinhalt als Float nach links setzen und dabei genug Platz für die Seitenleiste lassen, damit diese daneben fließen kann (oder als Float eingesetzt wird).
- Die Seitenleiste rechts vom Hauptinhalt absolut positionieren – das wäre dann links im normalen Fluss.

Wir entscheiden uns dafür, die Seitenleiste zu positionieren. Wenn wir es so lösen, können wir sie beliebig verschieben, und das ist dann auch weniger fehleranfällig für merkwürdige Zwischenfälle (darüber mehr in »Lieber Positionieren als Floaten« am Ende dieses Abschnitts). Damit das hinzukriegen ist, müssen wir etwas Platz schaffen, damit sie erscheinen kann, sonst würde sie den Hauptinhalt teilweise verdecken. Der Inhalt könnte sowieso etwas Padding vertragen, damit dessen Inhalte von den Kanten des Browserfensters, aber auch vom Masthead ferngehalten werden.

```
#sitemast h1 a {color: rgb(20%,40%,20%); text-decoration: none;}
#main {padding: 2em 25% 3em 1.5em;}
</style>
```

Das `1.5em` Padding links wurde der Breite des linken Rahmens im Masthead entnommen. Indem beide den gleichen Wert erhalten, wird sich die linke sichtbare Kante des Inhalts an der linken sichtbaren Kante des Masthead-Texts ausrichten. Was die `25%` rechtes Padding angeht – diesen Wert haben wir gewählt, weil wir damit genug Platz für eine Seitenleiste lassen.

Wir wollen allerdings nicht, dass die Seitenleiste den gesamten Raum einnimmt – wäre das Fall, dann würden sich der Hauptinhalt und die Seitenleiste praktisch berühren. Wir werden die Seitenleiste in der oberen rechten Ecke absolut positionieren und ihr eine Breite von `17%` spendieren. Und das wiederum ist ein Wert, den wir Pi mal Daumen gewählt haben. Wir haben auch einen gepunkteten roten Rahmen eingefügt, damit wir die Kanten der Seitenleiste sehen können – schauen Sie es sich in Abbildung 9.7 an.

```
#main {padding: 2em 25% 3em 1.5em;}
.panel {position: absolute; right: 0; top: 0; width: 17%;
  border: 1px dotted red;}
</style>
```

ABBILDUNG 9.7

Die Seitenleiste wird in der oberen rechten Ecke positioniert.

Na ja, für die Seitenleiste ist rechts vom Hauptinhalt bestimmt genug Platz! Das echte Problem besteht natürlich darin, dass wir die Seitenleiste über dem Masthead positioniert haben. Die müssen wir nun nach unten verschieben, damit sie genau unter dem unteren Rahmen des Mastheads sitzt.

Wie weit mag das sein? Also, das `h1`-Element ist `font-size` auf `2em` eingestellt (im Vergleich zu Text in Normalgröße), was bedeutet, dass alles verdoppelt wird. Es hat 1,5 em oberes Padding, aber der Inhalt ist 0,75 em hoch (durch den Wert für `line-height`). Das sind zusammen 2,25 em und verdoppelt 4,5 em. Beim Masthead-`div` ist auch ein em Padding unten, das sind also insgesamt 5,5 em.

```
.panel {position: absolute; right: 0; top: 5.5em; width: 17%;
  border: 1px dotted red;}
```

Durch diese Änderung wird dann der obere Rahmen der Seitenleiste den unteren Rahmen des Mastheads überlappen. Warum? Weil dieser eine Pixel für den unteren Rahmen unter den gerade von uns eingefügten 5,5 em platziert wird. Also liegt beim Masthead die obere Kante des unteren Rahmens 5,5 em unter dem oberen Rand des Dokuments, und die obere Rahmenkante der Seitenleiste ist exakt auf der gleichen Stelle. Das ist sogar eine prima Sache, weil wir in diesem Fall wollen, dass die beiden Rahmen überlappen.

Zur Positionierung Ränder verwenden

Wenn wir vermeiden wollten, dass die Rahmen überlappen, könnten wir die Seitenleiste um einen Pixel nach unten schieben (oder bei Bedarf auch mehr), indem wir einen oberen Rand setzen.

Jetzt ist ein guter Moment, um auch die Listeneinrückungen loszuwerden. Alle sollen nicht rausfliegen; es ist auch eine gute Sache, etwas von der vertikalen Separierung, die durch Padding und Ränder oben und unten geschaffen wird, zu erhalten. Zusätzlich sollte auch die Liste unter »Eric« unter Berücksichtigung der anderen Links eingerückt werden. Also lassen wir den Listen der obersten Ebene einen unteren Rand und etwas Padding oben und unten. Bei den verschachtelten Listen werden wir alles bis auf etwas linkes Padding entfernen. Dies wird in Abbildung 9.8 gezeigt.

```
.panel {position: absolute; right: 0; top: 5.5em; width: 17%;
  border: 1px dotted red;}
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;}
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;}
</style>
```

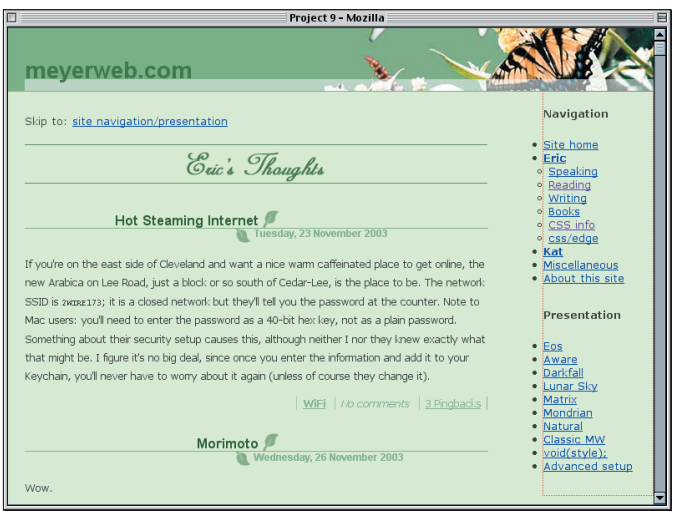


ABBILDUNG 9.8

Die neu positionierte Seitenleiste mit ihren geringfügig neu gestylten Listen

Einigen von Ihnen ist vielleicht aufgefallen, dass im IE 6 keine Listenpunkte (Bullets) auftauchen. Meines Erachtens liegt das daran, dass der Internet Explorer alles abschneidet, was aus dem positionierten Element herausragt. Wir könnten das darüber reparieren, dass wir für die Seitenleiste explizit den Wert für `overflow` ändern; allerdings wollen wir die Bullets sowieso loswerden, also wäre das ziemlich blöd.

Lieber Positionieren als Floaten

Es ist eine Erklärung wert, warum wir die Seitenleiste positioniert haben, anstatt den Hauptinhalt zu floaten und die Seitenleiste im normalen Fluss zu belassen. Um einen Float zu verwenden, hätten wir Styles wie folgt schreiben können:

```
#main {float: left; width: 75%;}
.panel {margin-left: 75%;}
```

Natürlich sind das vereinfachte Versionen dessen, was wir wirklich geschrieben hätten, aber so wird das Wesentliche der Methode deutlich. Damit wird der Inhalt auf die linke und die Seitenleiste auf die rechte Seite gestellt.

Warum wir das aber nicht so lösen, liegt hauptsächlich an den Bugs, die in der Rendering-Engine des Internet Explorers für Windows lauern, der sich nicht daran gewöhnen kann, im normalen Fluss ein Element über den Rahmen (oder das Padding) eines folgenden Elements zu floaten. Diese Bugs sind nicht sonderlich verlässlich, aber sie existieren und können einem ernsthaft das Layout einer Seite zerschießen.

Bei der anderen Float-Methode werden sowohl Inhalt als auch Seitenleiste gefloatet, also hieße das etwa:

```
#main {float: left; width: 75%;}
.panel {float: right; width: 25%;}
```

Darüber ist der Explorer eindeutig glücklicher, und das geht auch allen anderen modernen Browsern so. Der Nachteil liegt in der Schwierigkeit, Padding oder Ränder für diese Elemente einzustellen, die auf etwas anderem basieren als auf Prozentangaben. Wenn wir dem `div`*-Element des Hauptinhalts wieder seine 1,5 em Padding links zurückgeben, würden die beiden* `div`*s nicht mehr nebeneinander passen, weil das linke Padding zur Breite des* `div`*s dazugezählt wird, und das führt dann zu einer Element-Box mit einer Breite von (75 % + 1,5 em).*

Wir könnten mit den Zahlen herumspielen, bis wir etwas gefunden haben, das bei den meisten Leuten funktioniert, aber es wäre immer noch eine wackelige Lösung. Die einzige vernünftige Alternative wäre, nur Prozentangaben zu verwenden – also wie folgt:

```
#main {float: left; width: 70%; padding: 2em 0 3em 5%;}
.panel {float: right; width: 25%;}
```

Wenn Ihre Layoutbedingungen die Verwendung von Prozentangaben zulassen, ist das ausgezeichnet. Dann funktionieren die Floats bei Ihnen. Falls nicht, erleichtert Ihnen Positionieren das Leben deutlich.

9.4.4 Einfache Styles für die Seitenleiste

Die Bullets sind nun wirklich überfällig. Aber die werden wir leicht los.

```
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;
  list-style: none;}
```

Damit werden die Bullets aus allen `ul`-Elementen entfernt, die unterhalb vom `div`-Element der Seitenleiste liegen (weil sie die Klasse `panel` hat), egal wie tief sie in anderen Listen verschachtelt sind.

Bevor wir weitermachen, lassen Sie uns den in der Seitenleiste verwendeten Font ändern. Die Titel des Weblogs sind alle in Arial und auch der Masthead der Website, also sollten wir auch in der Seitenleiste Arial nehmen. Falls Arial aus irgendeinem Grund nicht verfügbar ist, geben wir als Plan B Verdana an

```
.panel {position: absolute; right: 0; top: 5.5em; width: 17%;
  font-family: Arial, Verdana, sans-serif;
  border: 1px dotted red;}
```

Dann gibt es da noch die `h4`-Elemente, die mit den Worten »Navigation« und »Presentation«. Wir sollten deren Ränder streichen, als Hintergrund einen dunkleren Grünton nehmen und zur besseren optischen Gefälligkeit etwas Padding hinzugeben – siehe Abbildung 9.9. Wir werden `font-size` auch auf 1 em setzen, damit der Text nicht größer oder kleiner wird als der Rest.

```
.panel {position: absolute; right: 0; top: 5.5em; width: 17%;
  font-family: Arial, Verdana, sans-serif;
  border: 1px dotted red;}
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%);
  font-size: 1em;}
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;
  list-style: none;}
```

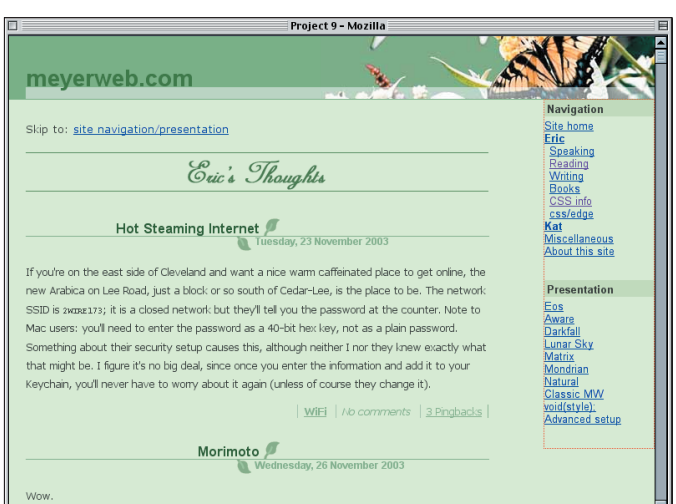


ABBILDUNG 9.9

Der Keim einer besser aussehenden Seitenleiste

Wir machen mit den h4-Elementen weiter. Statt Schwarz auf Grün wählen wir lieber ein Dunkelgrün auf Grün.

```
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%);
  font-size: 1em;}
  color: rgb(20%,40%,20%);}
```

Durch Hinzufügen von Rahmen heben sich die Überschriften mehr ab, aber einfach nur solide Rahmen zu nehmen ist schon recht einfallslos. Um sie visuell ansprechender zu gestalten, werden wir den unteren Rahmen gepunktet und die Rahmen der linken Kante etwas breiter machen und allen die gleiche Farbe geben. Das bekommen wir auf verschiedenen Wegen hin, aber wir versuchen es mal so, dass wir jedem Aspekt der Rahmen – Style, Breite und Farbe – eine eigene Regel geben.

```
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%);
  font-size: 1em;
  border-style: solid solid dotted; border-width: 1px 0 1px 2px;
  border-color: rgb(40%,60%,40%); color: rgb(20%,40%,20%);}
```

Nicht schlecht, aber wir können noch einiges mehr reißen! Die Verwendung von Kursivschrift könnte alles etwas interessanter machen und sie auch vom Rest des Textes der Seitenleiste abheben, also entscheiden wir uns dafür. Wir können auch die line-height reduzieren, so wie wir das beim Masthead gemacht haben, damit die Unterlängen über den unteren Rahmen des Elements selbst hinausragen.

```
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%);
  font-size: 1em; font-style: italic; line-height: 0.7em;
  border-style: solid solid dotted; border-width: 1px 0 1px 2px;
  border-color: rgb(40%,60%,40%); color: rgb(20%,40%,20%);}
```

Das wird bei »Navigation« den gewünschten Effekt erzielen, aber »Presentation« hat ja gar keinen Buchstaben mit Unterlänge. Das wäre allerdings anders, wenn wir alle Buchstaben auf Kleinbuchstaben umstellen; aus dem »P« würde dann ein »p«. Und diese Änderung wollen wir nun ausprobieren und obendrein die Laufweite erhöhen, damit ein luftigerer Eindruck entsteht. Das Ergebnis sehen wir in Abbildung 9.10.

```
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%);
  font-size: 1em; font-style: italic; line-height: 0.7em;
  text-transform: lowercase; letter-spacing: 1px;
  border-style: solid solid dotted; border-width: 1px 0 1px 2px;
  border-color: rgb(40%,60%,40%); color: rgb(20%,40%,20%);}
```

Reduzierte Zeilenhöhe

Wenn Sie sich dafür entscheiden, die line-height bei einem Element zu reduzieren, um einen Rahmen entlang der »Grundlinie« einzusetzen, wie wir das hier gemacht haben, wird der genaue Betrag für line-height vom verwendeten Font abhängen. Meist muss man etwas herumexperimentieren, um den richtigen Wert zu finden.

Abbildung 9.10

Überschriften auf Hochglanz poliert

9.4.5 Ein weiteres Blatt sprießt

Bisher war die Geschichte fast schon zu einfach, also leisten wir uns eine kleine Herausforderung. Ist Ihnen der freie Raum rechts von den Links in der Seitenleiste aufgefallen? Den wollen wir mal füllen. Und zwar mit etwas, was für das Auge gefällig ist, aber nicht über Gebühr ablenkt.

Eine kleine Grafik wäre da eine gute Wahl. Das ist der leichte Teil. Um es interessanter zu gestalten, sollte dieses Bild auf dem unteren Rahmen der Überschriften sitzen, damit das Bild in der Überschrift beginnt und sich bis unter den unteren Rahmen erstreckt – jedenfalls von der Optik her.

Dafür werden wir das in Abbildung 9.11 gezeigte Bild nehmen. (Hier erscheint es mit 500 % der Normalgröße.)

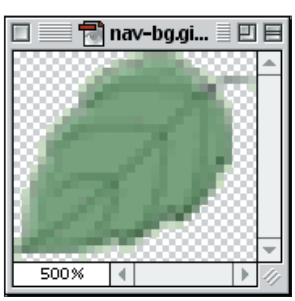


Abbildung 9.11

Das in der Seitenleiste verwendete Blattbild

Zuerst werden wir das Blatt-Bild in den Hintergrund der Überschriften setzen. Wir wollen, dass es in der Element-Box ganz rechts und ein Stück nach unten sitzt, damit seine Oberkante etwa in der Mitte der Überschrift beginnt. Das obere Padding plus die

Höhe des Inhalts beträgt 1,03 em, und es gibt da noch ein weiteres Pixel im unteren Padding. Also schätzen wir und sagen etwa 0,66 em.

```
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%) url(nav-bg.gif) 100% 0.66em no-repeat;
  font-size: 1em; font-style: italic; line-height: 0.7em;
  text-transform: lowercase; letter-spacing: 1px;
  border-style: solid solid dotted; border-width: 1px 0 1px 2px;
  border-color: rgb(40%,60%,40%); color: rgb(20%,40%,20%);}
```

Damit haben wir uns um den oberen Teil des Blatteffekts gekümmert. Jetzt geht es darum, dass die Blätter über den unteren Rahmen noch weitergehen. Zum Glück haben wir bereits fertige Elemente, die für diesen Zweck geeignet sind. Wir werden einfach das gleiche Bild in den Hintergrund der unsortierten Listen setzen und es nach oben schieben, damit beide aufeinander treffen.

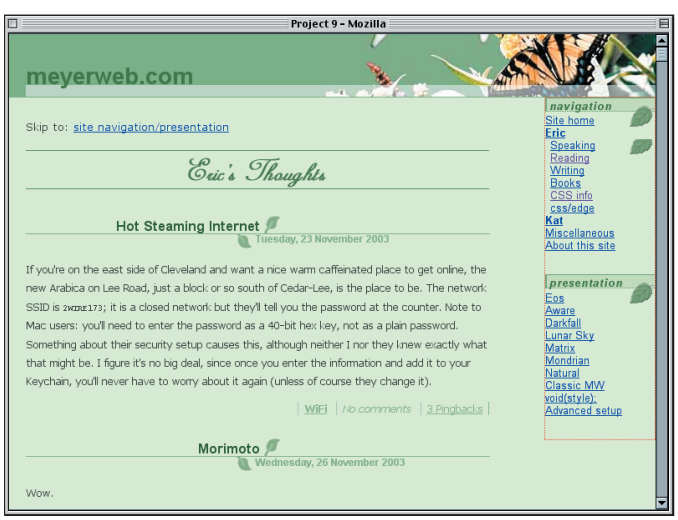
Wie hoch soll es geschoben werden? Nun, wir wissen, dass die Bildoberkante 0,37 em über dem unteren Bereich der unteren Kante des Überschrifteninhalts ist. Und es gibt noch Rahmen oben und unten, die jeweils einen Pixel haben, und noch das Padding unten mit einem weiteren Pixel. Also raten wir jetzt wieder und entscheiden uns dafür, dass es so um 0,5 em sein muss.

```
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;
  list-style: none;
  background: url(nav-bg.gif) 100% -0.5em no-repeat;}
```

Die Ergebnisse finden Sie in Abbildung 9.12.

ABBILDUNG 9.12

An der Seitenleiste sprießt es.



Coole Sache, das – bis auf etwas ganz Merkwürdiges: Da ist noch ein Stück Blatt! Das liegt daran, dass wir das Hintergrundbild auf alle `ul`-Elemente der Seitenleiste angewendet haben, und die verschachtelte Liste ist da mit eingeschlossen. Das müssen wir loswerden und fügen daher der `.panel ul ul`-Regel eine Deklaration hinzu.

```
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;
  background: none;}
```

Da wir gerade dabei sind, wollen wir den Text in dieser verschachtelten Liste auch auf kursiv stellen.

```
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;
  background: none; font-style: italic;}
```

9.4.6 Links in der Seitenleiste

Nun wollen wir uns um die eigentlichen Links kümmern. Der Text der Links ist schon viel zu lange eingezwängt gewesen, und da ist ein bisschen Padding für die Listenelemente genau das Richtige, damit sie wieder mehr Luft kriegen. Mit Padding oben und unten können wir sie auseinander ziehen, und durch Padding links rücken wir den Text ein Stück von der Kante der Seitenleiste ein.

```
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;
  list-style: none;
  background: url(nav-bg.gif) 100% -0.5em no-repeat;}
.panel ul li {padding: 0.15em 0 0.1em 0.5em;}
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;
  background: none; font-style: italic;}
```

Wir wollen allerdings nicht genau die gleichen Regeln für die Abstände auf die verschachtelte Liste anwenden. Um diese verschachtelten Links etwas mehr zu straffen, werden wir den Listenelementen das obere Padding abzwacken.

```
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;
  background: none; font-style: italic;}
.panel ul ul li {padding-top: 0;}
</style>
```

Nun zu den Farben: Blau und Purpurfarben, die Standardfarben für einen besuchten und einen unbesuchten Link, passen nicht sonderlich zu dem hier dominierenden grünen Thema. Darum lassen Sie uns ein paar Grundfarben für diese Link-Styles definieren und den unbesuchten Links ein Dunkelgrün geben und den besuchten ein Hellgrün. Das sehen Sie in Abbildung 9.13.

```
.panel ul ul li {padding-top: 0;}
.panel a:link {color: rgb(30%,50%,30%);}
.panel a:visited {color: rgb(50%,60%,50%);}
</style>
```

Schnitt bei Opera

Okay, eine Eigenartigkeit gibt es noch: Opera malt das Bild nicht unter dem Rahmen, also geht ein Schnitt quer durch das Blatt. Hätten wir einen soliden Rahmen genommen, wäre das kein Problem gewesen. Für den Rest des Projekts werden wir uns an den gepunkteten Rahmen halten, aber die User von Opera sollten ihn wohl solide machen.

ABBILDUNG 9.13
Verbesserte Präsentation der Links

Wir können das alles noch weiter treiben, indem wir die Links der Präsentation anders gestalten als die der Navigation. Weil die Optionen bei der Präsentation nicht wirklich mit der Navigation zu tun haben, sollten sie wohl auch nicht wie unbesuchte Links ausschauen. Das Einfachste wäre, sie wie besuchte Links aussehen zu lassen.

```
.panel a:visited, #presolinks a {color: rgb(50%,60%,50%);}
```

So werden die Farben der Links der Präsentation einheitlich gemacht, egal ob ein Browser annimmt, sie seien besucht oder nicht.

Wo wir gerade über die Farben der Seitenleisten-Links sprechen, könnte ein Rollover-Effekt ganz passend sein. Den wollen wir allen Navigations-Links zuweisen, aber nicht den Präsentations-Links. (Warum? Einfach darum.) Also werden wir eine neue Regel einfügen, die alle Links der Navigation, über denen der Mauszeiger ruht, eine dunkelbraune Farbe wie Holz geben.

```
.panel a:visited, #presolinks a {color: rgb(50%,60%,50%);}
.panel a:hover {color: rgb(50%,30%,20%);}
</style>
```

Diese Regel wirkt sich nicht auf die Präsentations-Links aus, weil die Spezifität von `#presolinks a` höher ist als die von `.panel a:hover`. Darum wird die *hover*-Farbe gegen die Farbe verlieren, die über `#presolinks a` zugewiesen wird. In diesem Fall ist die Reihenfolge der Regeln nicht von Bedeutung – dank der Unterschiede in der Spezifität.

Spezifität der Links

Tatsächlich ist die Spezifität genau der Grund für die Empfehlung, dass Link-Styles in dieser Reihenfolge sein sollen: `link` (für noch nicht besuchte Verweisziele), `visited` (für bereits besuchte Verweisziele), `hover` (für Verweise, während der Anwender mit der Maus darüber fährt), `active` (für angeklickte Verweise). Mehr können Sie unter `http://www.meyerweb.com/eric/css/link-specificity.html` lesen.

9.4.7 Änderungen am Rahmenwerk

Nun wird es langsam Zeit, dass wir diesen gepunkteten roten Rahmen entfernen – der ist auch schon längst überfällig. Also werfen wir diese Deklaration aus der `.panel`-Regel raus.

```
.panel {position: absolute; right: 0; top: 5.5em; width: 17%;
  font-family: Arial, Verdana, sans-serif;}
```

Diese Änderung heißt, dass der obere Rahmen der »Navigation«-Überschrift nun den unteren Rahmen des Mastheads überlappen wird, und genau das hat der gepunktete rote Rahmen auch gemacht, bevor wir ihn entfernt haben.

Nachdem wir den Rahmen der Seitenleiste entfernt haben, sollten wir allerdings wieder eine Form der optischen Trennung zwischen den Links der Seitenleiste und dem Hauptinhalt herstellen. Anstatt einen linken Rahmen einzuführen, wollen wir stattdessen den unsortierten Listen, die die Links enthalten, den Rahmen links geben.

```
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;
  list-style: none; border-left: 1px solid rgb(45%,65%,45%);
  background: url(nav-bg.gif) 100% -0.5em no-repeat;}
```

Erinnern Sie sich noch an das Blatt, wie es bei der verschachtelten Liste ganz oben auftauchte? Wir müssen ein neuerliches Auftreten dieser Wiederholung aus der verschachtelten Liste verhindern, indem wir allen verschachtelten Listen den linken Rahmen nehmen. Dies sehen Sie in Abbildung 9.14.

```
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;
  background: none; border-left: none; font-style: italic;}
```

ABBILDUNG 9.14

Rahmen für die Listen verstärken die visuelle Separierung.

9.4.8 Natürliche Hervorhebung

Wir sind fast fertig, aber einige kleinere Sachen müssen wir noch erledigen. Eins davon ist die Erfüllung unseres dritten Projektziels: den Namen dieses Themas irgendwie hervorzuheben. Der Name dieses Themas ist passend genug: »Natural«. Wenn wir uns das Markup für diesen Link anschauen, können wir eine einfache Möglichkeit sehen, den Namen des Themas zu akzentuieren.

```
<li id="natural"><a href="#"
  onclick="setActiveStyleSheet('Natural'); return false;"
  title="Wildlife and greenery">Natural</a></li>
```

Diese `id` gibt uns alles, was wir brauchen, um den Namen des Themas anders als die der anderen zu gestalten. Als Anfang wollen wir gleich mal die Link-Farbe auf Dunkelgrün setzen.

```
.panel ul ul li {padding-top: 0;}
.panel #natural a {color: rgb(30%,40%,30%);}
.panel a:link {color: rgb(30%,50%,30%);}
```

Das ist erledigt, und nun können wir den Text auf kursiv und fett setzen, indem wir das Listenelement `#natural` direkt stylen. Diese neuen Font-Styles werden durch den Link vererbt, die Farbe allerdings nicht, weil wir ihm gerade explizit eine Farbe zugewiesen haben.

```
.panel ul ul li {padding-top: 0;}
.panel #natural {font-weight: bold; font-style: italic;}
.panel #natural a {color: rgb(30%,40%,30%);}
```

Genauer betrachtet sieht der Link nun beinahe wie die Überschrift aus, also werden wir noch ein paar von den Styles mit ins Boot holen, die wir für die Überschriften benutzt haben: den erhöhten Buchstabenzwischenraum und den Text in Kleinbuchstaben.

```
.panel #natural {font-weight: bold; font-style: italic;
  letter-spacing: 1px; text-transform: lowercase;}
```

Auf die gleiche Weise fügen wir bei diesem Listenelement einen gepunkteten Rahmen ein, um ihn von den anderen abzusetzen. Wir wollen nicht, dass er an der rechten Kante verschwindet, also werden wir einfach einen allgemeinen Rahmen definieren und ihn dann auf der rechten Seite abschalten. Eine Hintergrundfarbe, die einen Tick dunkler ist als der Seitenhintergrund, werden wir auch einfügen.

```
.panel #natural {font-weight: bold; font-style: italic;
  letter-spacing: 1px; text-transform: lowercase;
  border: 1px dotted rgb(45%,65%,45%); border-right: none;
  background: rgb(83%,90%,78%);}
```

Schauen wir uns das Aussehen des »Natural«-Links an, unterscheidet er sich definitiv von seinen Nachbarn. Da ist nur ein kleines Detail – der linke Rahmen. Er sitzt genau im Rahmen der Liste, und das wirkt nicht besonders schön. Wir könnten ihn entfernen, aber wir haben eine bessere Idee: Wir könnten ihn doch den Rahmen der Liste überlappen lassen – dadurch schneidet er in diesen Rahmen ein Loch, und der Unterschied zwischen dem Namen des Themas und dem Rest der Seite verschwimmt visuell. Dafür braucht es nur einen negativen linken Rand von einem Pixel bei dem Listenelement, und das Ergebnis sehen wir in Abbildung 9.15.

```
.panel #natural {font-weight: bold; font-style: italic;
  letter-spacing: 1px; text-transform: lowercase;
  border: 1px dotted rgb(45%,65%,45%); border-right: none;
  background: rgb(83%,90%,78%); margin-left: -1px;}
```

Abbildung 9.15

Der Name des aktuellen Themas wird hervorgehoben.

9.4.9 Der letzte Feinschliff

Ein paar Sachen sind nun noch übrig geblieben. Als Erstes brauchen wir den Skip-Link eigentlich nicht mehr dort zu lassen. Wir können verschiedene Methoden nutzen, um ihn vom Bildschirm zu entfernen, u.a. auch ihn einfach auf `display: none` zu setzen. Das wollen wir dann auch machen, weil es immerhin das Einfachste ist. Diesen Style haben wir nur deswegen oben im Stylesheet eingefügt, um ihn möglichst bald aus dem Weg zu schaffen.

```
@import url(project08.css);
.skipper {display: none;}
#sitemast {margin: 0; padding: 0 0 1em;
  border: 1px solid rgb(45%,65%,45%); border-width: 0 0 1px 1.5em;
  background: rgb(73%,82%,73%) url(mh-light.jpg) 100% -30px no-repeat;}
```

Wir könnten auch noch etwas an den Farben der Links auf der Seite insgesamt drehen anstatt bloß bei der Seitenleiste. Hierbei entscheiden wir uns bei den unbesuchten Links für ein helles Mittelgrün und bei den besuchten für ein blasseres Grün. Das wird sich auf alle Links auswirken, die nicht über andere Wege gestylt wurden, und das bedeutet, die Regel wird sich weder auf die Links der Seitenleiste noch auf die des Weblogs auswirken, die wir im vorigen Kapitel gestylt haben.

Die Regeln, die sich um die neuen Farbeffekte für die Links kümmern, haben wir nur aus dem Grund ans Ende des Stylesheets gesetzt, dass nichts dagegen spricht.

```
.panel a:hover {color: rgb(50%,30%,20%);}
a:link {color: rgb(0%,50%,40%); background: transparent;}
a:visited {color: rgb(30%,50%,30%); background: transparent;}
</style>
```

Weg mit der Zugänglichkeit

Obwohl die Skip-Links aus Gründen der besseren Zugänglichkeit eingefügt wurden, werden die meisten Vorlesegeräte sie nicht sehen, wenn sie auf `display: none` eingestellt werden. Das Gleiche gilt für praktisch jede andere Methode, um Skip-Links aus Ihrem Web-Design zu entfernen. Bis diese kaputten Vorlesegeräte richtig damit umgehen können, wird auf nicht angezeigte Skip-Links wahrscheinlich nicht zugegriffen werden können.

Warnungen aus dem Hintergrund

Wir haben für die Links explizit `transparent`-Hintergründe genommen, um zu verhindern, dass bei den meisten CSS-Validatoren Warnungen ausgelöst werden. Beachten Sie, dass eine Warnung nicht den Erfolg der Validierung verhindert – das kann nur ein Fehler.

Wie es aussieht, nachdem der Skip-Link weg ist und die Links des Dokuments farblich geändert sind, können Sie in Abbildung 9.16 bewundern.

ABBILDUNG 9.16

Entfernen des Skip-Links und Änderung der Link-Farben

Nun wollen wir den Footer am Ende der Seite stylen. Wir brauchen uns keine Sorgen darüber zu machen, dass er vom Text des Hauptinhalts getrennt bleibt, weil das über das untere Padding beim `div`-Element des Hauptinhalt selbst erledigt wird. Wir können uns also allein auf das Styling der Fußzeile konzentrieren. Wir beginnen mit einem doppelten Rahmen oben, einem auf Null gestellten Rand und Padding oben und unten bei der Fußzeile selbst.

```
a:visited {color: rgb(30%,50%,30%); background: transparent;}
#footer {border-top: 3px double; margin: 0; padding: 0.75em 1em 1em;}
</style>
```

Das Padding stellt sicher, dass der Inhalt der Fußzeile sich nicht gegen die Kanten des Browser-Fensters drängt. In die Fußzeile wird gewöhnlich der ganze langweilige Kram wie Copyright-Hinweise und anderer juristischer Schnickschnack gestopft, also halten wir die Textgröße gering.

```
#footer {border-top: 3px double; margin: 0; padding: 0.75em 1em 1em;
 font-size: 75%;}
```

Um die visuelle Trennung der Fußzeile abzuschließen, werden wir die Text- und die Hintergrundfarbe ändern. Diese Farben basieren auf Variationen des allgemeinen Seitenhintergrunds und des Seitenthemas. (Wenn Ihnen die nicht gefallen, können Sie auch gerne andere Werte wählen.)

```
#footer {border-top: 3px double; margin: 0; padding: 0.75em 1em 1em;
  font-size: 75%;
  color: rgb(20%,40%,20%); background: rgb(73%,82%,73%);}
```

Reduktion der Textgröße

Wir hätten auch einen Wert wie `smaller` nehmen können, um die Textgröße zu reduzieren. In solchen Situationen ist der exakt verwendete Wert oft eine Geschmackssache des Designers.

Uns bleibt nun noch eine einzige Sache, nämlich die Lücke zwischen den beiden Absätzen in der Fußzeile zu schließen. Indem wir den Absätzen die Ränder und das Padding nehmen, lassen wir sie wie zwei Zeilen im gleichen Absatz wirken – siehe Abbildung 9.17.

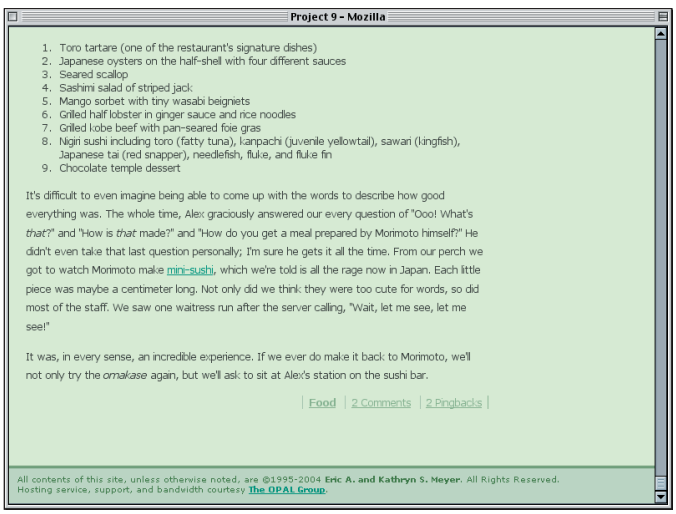


Abbildung 9.17

Auch die Fußzeile wird gestylt.

```
#footer {border-top: 3px double; margin: 0; padding: 0.75em 1em 1em;
  font-size: 75%;
  color: rgb(20%,40%,20%); background: rgb(73%,82%,73%);}
#footer p {margin: 0; padding: 0;}
</style>
```

Das Endergebnis unserer vielen Arbeit zeigen wir hier in Listing 9.2.

Listing 9.2 Das vollständige Stylesheet

```
@import url(project08.css);
.skipper {display: none;}
#sitemast {margin: 0; padding: 0 0 1em;
  border: 1px solid rgb(45%,65%,45%); border-width: 0 0 1px 1.5em;
  background: rgb(73%,82%,73%) url(mh-light.jpg) 100% -30px no-repeat;}
#sitemast h1 {margin: 0; padding: 1.5em 0.5em 0 0;
  font: bold 2em/0.75em Arial, Verdana, sans-serif;
  background: rgb(45%,65%,45%) url(masthead.jpg) 100% -30px no-repeat;}
#sitemast h1 a {color: rgb(20%,40%,20%); text-decoration: none;}
#main {padding: 2em 25% 3em 1.5em;}
.panel {position: absolute; right: 0; top: 5.5em; width: 17%;
  font-family: Arial, Verdana, sans-serif;}
.panel h4 {margin: 0; padding: 0.33em 0.5em 1px 0.25em;
  background: rgb(75%,85%,70%) url(nav-bg.gif) 100% 0.66em no-repeat;
  font-size: 1em; font-style: italic; line-height: 0.7em;
  text-transform: lowercase; letter-spacing: 1px;
  border-style: solid solid dotted; border-width: 1px 0 1px 2px;
```

```
  border-color: rgb(40%,60%,40%); color: rgb(20%,40%,20%);}
.panel ul {margin: 0 0 1.5em; padding: 0.25em 0 0.5em;
  list-style: none; border-left: 1px solid rgb(45%,65%,45%);
  background: url(nav-bg.gif) 100% -0.5em no-repeat;}
.panel ul li {padding: 0.15em 0 0.1em 0.5em;}
.panel ul ul {margin: 0; padding: 0 0 0 0.5em;
  background: none; border-left: none; font-style: italic;}
.panel ul ul li {padding-top: 0;}
.panel #natural {font-weight: bold; font-style: italic;
  letter-spacing: 1px; text-transform: lowercase;
  border: 1px dotted rgb(45%,65%,45%); border-right: none;
  background: rgb(83%,90%,78%); margin-left: -1px;}
.panel #natural a {color: rgb(30%,40%,30%);}
.panel a:link {color: rgb(30%,50%,30%);}
.panel a:visited, #presolinks a {color: rgb(50%,60%,50%);}
.panel a:hover {color: rgb(50%,30%,20%);}
a:link {color: rgb(0%,50%,40%); background: transparent;}
a:visited {color: rgb(30%,50%,30%); background: transparent;}
#footer {border-top: 3px double; margin: 0; padding: 0.75em 1em 1em;
  font-size: 75%;
  color: rgb(20%,40%,20%); background: rgb(73%,82%,73%);}
#footer p {margin: 0; padding: 0;}
```

9.5 Spielwiese

1. Konvertieren Sie das Design, um für die beiden Spalten Floats zu verwenden, wie wir es im Abschnitt »Lieber Positionieren als Floaten« besprochen haben. Beachten Sie, dass die in diesem Abschnitt angegebenen Styles nicht ausreichend sein werden – Sie werden sich auch noch um Sachen wie Ränder, Padding und die Platzierung der Fußleiste zu kümmern haben.

2. Drehen Sie das Layout um, damit die Seitenleiste links steht und der Hauptinhalt rechts. Dafür werden Änderung der Rahmen an der Seitenleiste nötig sein, und bei der Textausrichtung muss sich wohl auch etwas ändern. Versuchen Sie auch, den Masthead so zu modifizieren, dass er visuell zu der geänderten Platzierung der Seitenleiste passt.

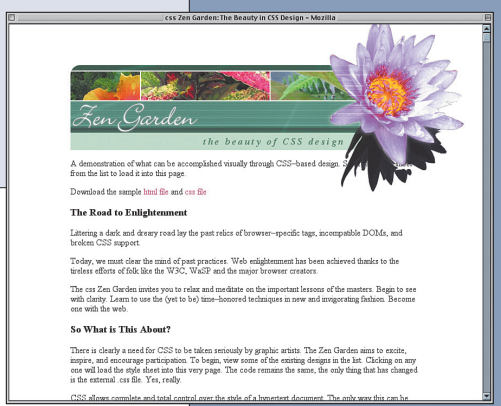
3. Und hier haben wir noch ein ganz ambitioniertes Vorhaben: Wandeln Sie die Seitenleiste unter Verwendung der Techniken, die wir in Projekt 6 erforscht haben, in eine horizontale Symbolleiste um – komplett mit Drop-down-Menüs. Wenn Sie es wirklich auf die Spitze treiben wollen, können Sie die Drop-down-Styles mit der »Sliding Doors«-Technik aus dem Projekt 7 »Hereinspaziert! Attraktive Tabs für Ihre Website« kombinieren, um eine wirklich hervorragende Symbolleiste zu schaffen.

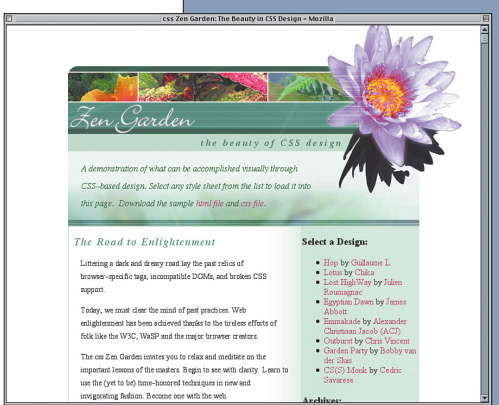
10

Design im Garten

I love you twisted
And I love you straight
I'd write it down but I can't concentrate
Words won't me obey they do as they please
And all I am left with is these

Elvis Costello





Im Mai des Jahres 2003 erfuhr die CSS-Gemeinschaft von einer erstaunlichen neuen Ressource: dem CSS Zen Garden. Auf dieser Site wird ein HTML-Dokument bereitgestellt, das nicht geändert werden kann, und alle Designer sind aufgefordert, ein Stylesheet zu schreiben, das dieses Dokument auf originelle Weise, stilistisch elegant und visuell ansprechend präsentiert.

Zum Zeitpunkt dieses Schreibens gibt es nahezu einhundert verschiedene offizielle Designs für den Zen Garden, jedes auf seine Weise beeindruckend und jedes völlig anders – einige radikal anders. Diese Site soll zeigen, wie wunderschön CSS-Design sein kann und dass diese Schönheit basierend auf einer relativ einfachen Dokumentstruktur aufgebaut und das gleiche Dokument auf völlig unterschiedliche Arten präsentiert werden kann. Das ist dieser Site in jeglicher Hinsicht auf brillante Weise gelungen.

Da ich *Eric Meyer on CSS* mit einem sehr anspruchsvollen Projekt abgeschlossen habe (es ging damals um die Neugestaltung des Buchlayouts mit CSS), ist es nahe liegend, eine ähnliche Herausforderung an das Ende dieses Buches zu stellen. Und

es erscheint mir sehr passend, dass diese darin bestehen soll, ein Design für den Zen Garden zu schaffen.

Da gab es nur ein Problem. Ich will ehrlich zugeben, dass ich kein toller visueller Künstler bin (und ich höre schon welche sagen, das sei eine freche Untertreibung), doch die Designs für den Zen Garden müssen wirklich hervorragend aussehen. Also beschloss ich, mich an jemanden zu wenden, der ein hervorragender visueller Künstler ist: Dave Shea, der Erfinder des CSS Zen Gardens und einer der Technischen Berater für genau dieses Buch. Dave hat ein wunderschönes Design geschaffen, und ich habe mich daran gemacht, es in ein CSS-Layout zu verwandeln. Und in diesem Projekt werde wir alle Schritte nachvollziehen, die ich dafür gemacht habe.

10.1 Projektziele

Dieses Mal stellt sich uns wirklich nur ein Projektziel: Wir erhalten ein visuelles Design und konvertieren es zu einem CSS-Layout. Dieses schlichte Ziel gliedert sich allerdings in drei speziellere Ziele, insbesondere da dieses Design für den CSS Zen Garden bestimmt ist.

- Wir wollen ein Layout kreieren, ohne im uns zur Verfügung stehenden HTML-Dokument ein einziges Zeichen bei Inhalt oder Markup zu ändern. Uns steht nur das CSS zur Verfügung.
- Das Layout soll in den folgenden Browsern gut aussehen: Internet Explorer 5.5+/Win, Internet Explorer 5/Mac, Safari und die aktuellen Gecko-basierten Browser (zum Zeitpunkt dieses Schreibens sind das Mozilla 1.6 und Firefox 0.8).
- Das Design soll Änderungen der Schriftgröße tolerieren, und zwar bis zu 150 % über der Standard-Textgröße der User.

Wir werden hier also zwei recht verbreitete Situationen im Leben eines Webdesigners kombinieren. Die erste besteht darin, ein von einem Grafiker geschaffenes Layout zu reproduzieren; das kommt oft in großen Firmen (oder jeder anderen Organisation) vor, wo das Aussehen einer Site nicht den Programmierern, sondern den Grafikern obliegt. Die zweite Situation ist, dass Sie ein Markup stylen sollen, das Sie nicht ändern dürfen. In diesem speziellen Fall ist das Markup so eingerichtet, dass ein CSS-Layout damit erleichtert wird – das ist eine gute Sache.

10.2 Vorbereitungen

Schauen Sie in der Einführung nach, wie Dateien von der Website heruntergeladen werden können.

Laden Sie die Dateien für Projekt 10 von der Website dieses Buchs herunter. Wenn Sie die Schritte selber nachvollziehen wollen, laden Sie die Datei `ch10proj.html` in den Editor Ihrer Wahl. Diese Datei werden Sie im Verlauf des Kapitels bearbeiten, abspeichern und neu laden.

10.3 Die Grundlagen

Wie immer wollen wir uns erst die Dokumentstruktur anschauen, bevor wir sie mit CSS nachbilden. Die grundlegende Struktur des Dokuments finden Sie in Listing 10.1.

Listing 10.1 Die Dokumentstruktur

```
<body id="css-zen-garden">
<div id="container">
  <div id="intro">
    <div id="pageHeader">
      <h1><span>css Zen Garden</span></h1>
      <h2><span>The Beauty of <acronym
       title="Cascading Style Sheets">CSS</acronym> Design</span></h2>
    </div>
    <div id="quickSummary">
      [...Inhalt...]
    </div>
    <div id="preamble">
      <h3><span>The Road to Enlightenment</span></h3>
      [...Inhalt...]
    </div>
  </div>
  <div id="supportingText">
    <div id="explanation">
      <h3><span>So What is This About?</span></h3>
      [...Inhalt...]
    </div>
    <div id="participation">
      <h3><span>Participation</span></h3>
      [...Inhalt...]
    </div>
    <div id="benefits">
      <h3><span>Benefits</span></h3>
      [...Inhalt...]
    </div>
    <div id="requirements">
      <h3><span>Requirements</span></h3>
      [...Inhalt...]
    </div>
    <div id="footer">
      [...Inhalt...]
    </div>
  </div>
  <div id="linkList">
    <div id="linkList2">
      <div id="lselect">
        <h3 class="select"><span>Select a Design:</span></h3>
        [...Linkliste...]
      </div>
      <div id="larchives">
```

Extra-divs?

Die am Ende des Listing 10.1 erwähnten »`[... Extra-divs...]`« sind `divs` ohne Inhalte, die mit eingefügt wurden, damit dem Designer bei Bedarf Extra-Elemente zur Verfügung stehen. Wir brauchen sie nicht, also werden wir sie auch nicht mehr erwähnen.

```
          <h3 class="archives"><span>Archives:</span></h3>
          [...Linkliste...]
        </div>
        <div id="lresources">
          <h3 class="resources"><span>Resources:</span></h3>
          [...Linkliste...]
        </div>
      </div>
    </div>
  </div>
  [...Extra-divs...]
  </body>
```

In der Dokumentstruktur steckt noch eine Menge mehr, aber das Listing 10.1 deckt im Prinzip den Teil ab, den wir für dieses Projekt brauchen. Das Listing 10.1 zeigt nur das absolute Minimum, das Sie brauchen, um den Rest des Projekts zu verstehen, weil ein vollständiges Listing des HTML-Dokuments über etliche Seiten ginge, und diese schicken farbigen Bücher werden umso teurer, je mehr Seiten sie haben.

Nun wollen wir uns einmal das Design ansehen, das wir nachbauen wollen. Es ist die in Abbildung 10.1 gezeigte Photoshop-Datei.

Das werden wir also durch CSS gestalten. Damit alles funktioniert, werden wir eine Reihe von Bildern verwenden, die später nützlich sein werden. Das sind die Hintergründe und Designelemente, denen wir das gleiche visuelle Aussehen geben müssen wie in Abbildung 10.1. Im Einzelnen handelt es sich um folgende Bilder:

- Die Kopfzeile mit dem Skript-Font »Zen Garden«
- Die Blume auf der rechten Seite der Kopfzeile
- Der verschwommene hellgrüne Hintergrund des Bereichs »quick summary« (Zusammenfassung)
- Die verblichenen Blumen unten in der Spalte mit dem Haupttext
- Der Hintergrund der Seitenleiste und die gestufte Abtönung auf der rechten Seite der Haupttextspalte
- Der Hintergrund des Footers
- Die drei Überschriften der Seitenleiste (»Design List« usw.)

Wir werden uns alle genauer anschauen, wenn wir sie jeweils benötigen. So, nun machen wir uns ans Styling!

Rechtshinweis

Die in diesem Design verwendeten Fotografien sind Copyright © 2003 Eric A. Meyer und werden mit seiner Erlaubnis verwendet. Die anderen in diesem Design verwendeten Bilder sind Copyright © 2004 Dave Shea und werden mit seiner freundlichen Genehmigung verwendet.

Abbildung 10.1

Die Referenzdatei, deren visuelles Design wir nachbauen (50 % der Normalgröße)

10.4 Das Design wird erstellt

Zu Beginn werden wir einige »globale« Styles einrichten: Regeln, die für das gesamte Dokument gelten. Diese beinhalten das Entfernen der Seitenränder (oder des Paddings), das Einstellen von grundlegenden Vorder- und Hintergrundfarben und die Entfernung der Rahmen und Unterstriche der `acronym`- bzw. der `link`-Elemente.

```
<style type="text/css" title="currentStyle">
body {margin: 0; padding: 0; text-align: center;
  color: #000; background: #FFF;}
acronym {border: none;}
a {text-decoration: none;}
</style>
```

Wie Sie sehen, haben wir auch den gesamten Text des Dokuments zentriert. Das haben wir gemacht, weil das Design mittig sein soll, und der beste Weg, um Elemente in

IE5.5/Win zu zentrieren, läuft über die Deklaration `text-align: center;`. Die CSS-Spezifikation besagt eindeutig, dass `text-align` keine Elemente zentrieren soll, aber der IE5.5/Win macht es doch. Das werden wir schon bald wieder ausgleichen, aber zuerst wollen wir einige Link-Styles definieren. Wir werden unbesuchten Links einen pinkfarbenen Ton geben, besuchten Links ein mattes Rot und alle Links, über denen der Mauszeiger steht, bekommen einen Unterstrich – siehe Abbildung 10.2.

```
a {text-decoration: none;}
a:link {color: rgb(179,63,96);}
a:visited {color: rgb(90,32,48);}
a:hover {text-decoration: underline;}
</style>
```

ABBILDUNG 10.2

Hier sind die ersten Styles schon angewendet.

Denken Sie daran, dass die Zentrierung des Textes ein temporärer Effekt ist, der nur deswegen vorgenommen wurde, um den Bug im IE5.5/Win umgehen zu können. Bereits im nächsten Abschnitt werden wir das überschreiben und den Text wieder nach links bringen, wo er auch hingehört.

10.4.1 Die Saat wird ausgebracht

Wenn Sie das Listing 10.1 noch einmal zu Rate ziehen, werden Sie sehen, dass der gesamte Seiteninhalt von einem `div`-Element mit dem Individualformat (`id`) `container` umschlossen wird. Darum wird unser erster Schritt beim eigentlichen Stylen des Inhalts darin liegen, einige grundlegende Parameter für diesen Container festzulegen.

Schauen wir uns noch einmal Abbildung 10.1 an. Die Breite der Hauptspalte (ohne die große Blume in der Ecke oben rechts) beträgt 647 Pixel. Wir wissen nicht, wie es zu dieser Breite gekommen ist – wir nehmen sie einfach so hin. Dementsprechend breit werden wir also den Container anlegen. Gleichzeitig werden wir das gesamte Padding entfernen und sowohl explizit einen oberen Rand einstellen als auch die Seitenränder auf `auto` stellen.

```
a:hover {text-decoration: underline;}
#container {width: 647px; margin: 75px auto 0; padding: 0;}
</style>
```

Der obere Rand von 75 Pixel ist nur ein Hilfsmaß, das wir bei Bedarf später noch anpassen können. Unser Plan ist, über der Kopfzeile (Masthead) genug Platz zu schaffen, damit dort die große Blüte hineinpasst.

Über die Randeinstellung `auto` wird der Container zentriert, aber nur, weil er eine explizite `width` hat. Bei CSS wird ein Blocklevel-Element, dem eine explizite `width` und automatische rechte und linke Ränder zugewiesen wurden, in seinem Container-Block zentriert. Der IE5.5/Win macht das nicht, aber er wird doch Elemente zentrieren, wenn deren Eltern auf `text-align: center;` gesetzt sind. Darum hatten wir den Text im `body`-Element zentriert, damit beim IE5.5/Win der Container zentriert wird, und wir haben die Ränder auf `auto` gestellt, damit der Container in aktuelleren Browsern zentriert wird.

Weil `text-align` eine vererbte Eigenschaft ist, wird Text innerhalb des Containers ebenfalls zentriert. Also werden wir diese Zentrierung innerhalb des Containers außer Kraft setzen, indem wir ihm einen anderen Wert für `text-align` geben.

```
#container {width: 647px; margin: 75px auto 0; padding: 0;
  text-align: left;}
```

Das wird den Text innerhalb des Containers wieder linksbündig ausrichten, ohne die Zentrierung im IE5.5/Win zu kippen.

10.4.2 Die Kopfzeile

Wenn wir nun den Container »hineingucken«, finden wir als Erstes die Kopfzeile (pageHeader), dessen Markup in Listing 10.2 aufgeführt ist.

Listing 10.2 Das Markup der Kopfzeile

```
<div id="pageHeader">
  <h1><span>css Zen Garden</span></h1>
  <h2><span>The Beauty of <acronym
   title="Cascading Style Sheets">CSS</acronym> Design</span></h2>
</div>
```

Festes und fließendes Design

Zum Thema Design mit fester Breite vs. »fließendes« Design (das auf der Breite des Browser-Fensters basiert) gibt es große Meinungsverschiedenheiten, und es ist nicht unser Anspruch, diese hier aufzulösen. Geht man von der vor uns liegenden Aufgabe und den uns zur Verfügung stehenden Bilddateien aus, ist ein Design mit fester Breite sinnvoller. Allerdings kann ein ähnliches Design auch ohne feste Breite geschaffen werden.

ABBILDUNG 10.3

Das für die Kopfzeile verwendete Bild

Wie sich herausstellt, ist das weitaus mehr, als wir für die Neuerstellung des Kopfzeilendesigns benötigen. Abbildung 10.3 zeigt uns eines der beiden Bilder, die wir in der Kopfzeile verwenden werden. Die Maße sind 477 Pixel breit und 157 Pixel hoch.

Klein/Großschreibung

HTML beachtet bei der Definition der Werte für `class` und `id` die Groß- und Kleinschreibung, also müssen wir `pageHeader` statt beispielsweise `pageheader` schreiben. Ältere Browser nehmen es damit nicht so genau, aber dafür die neueren Browser, also sollten Sie darauf achten, die Schreibweise einheitlich zu halten.

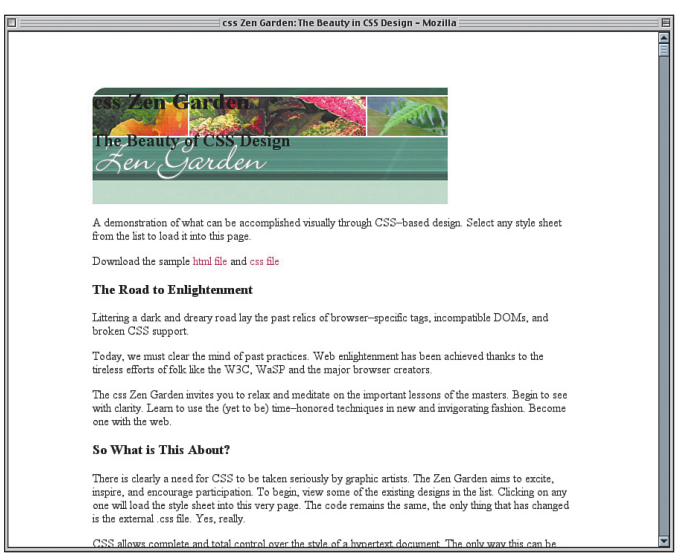
Wir brauchen dieses Bild nur in den Hintergrund des `pageHeader` zu setzen und sicherzustellen, dass das `div`-Element groß genug ist, um das ganze Bild zu zeigen. Das Bild soll auch nur einmal erscheinen, also achten wir darauf, dass es sich nicht wiederholt.

```
#container {width: 647px; margin: 75px auto 0; padding: 0;
  text-align: left;}
#pageHeader {background: url(pageHeader.jpg) 0 0 no-repeat;
  height: 157px; width: auto;}
</style>
```

Obwohl wir den Wert für die Höhe genau dem Bild entsprechend eingestellt haben, haben wir es bei der Breite bei einer automatischen Berechnung belassen. Das heißt, es wird so breit wie möglich sein, aber immer noch in sein Elternelement passen, und das ist der `container`. Also wissen wir, dass der `pageHeader` 647 Pixel breit sein wird, genau wie der `container`. Daher kommt es, dass in Abbildung 10.4 das Bild nicht so breit ist wie der nachfolgende Text.

ABBILDUNG 10.4

Der Hintergrund der Kopfzeile erhält ein Bild.



Das ist allerdings auch gar kein Problem, denn wir brauchen Platz, um unsere große dekorative Blume einsetzen zu können. Vielleicht fällt Ihnen auf, dass der Text der `h1`- und `h2`-Elemente immer noch vor dem Hintergrund sichtbar ist. Das ist auch zu erwarten gewesen. Aber keine Sorge, wir werden beide sinnvoll einsetzen.

Bevor wir weitermachen, sollten wir noch einen Schritt zurückgehen und etwas erledigen, was wir übersehen haben. Da wir für die Erstellung dieses Layouts nur CSS benutzen können, müssen wir verschiedene Elemente positionieren. Um sie alle mit Bezug auf den Container zu positionieren, muss dieser ohne Offset relativ positioniert werden.

```
#container {width: 647px; margin: 75px auto 0; padding: 0;
  text-align: left; position: relative;}
```

Das erstellt einen Container-Block für alle relativ positionierten Elemente innerhalb des Containers, ohne dass der `container` tatsächlich aus seiner Position im normalen Dokumentfluss fortbewegt wird.

10.4.3 Jetzt wird es blumig

Nun können wir das Bild der Blüte aus der Abbildung 10.5 platzieren. Es ist 250 Pixel breit und 333 Pixel hoch und verfügt über transparente Bereiche (diese werden in der Abbildung 10.5 durch das grau-weiße Schachbrettmuster angedeutet), durch den andere Hintergründe oder gar Inhalte sichtbar sind.

ABBILDUNG 10.5

Dieses Bild soll die Kopfzeile rechts schmücken.

Nun müssen wir nur noch herauskriegen, wie wir es an seinen Platz bringen. Wir können den Hintergrund des `pageHeader` nicht nehmen, weil der bereits ein Bild hat. Stattdessen werden wir das `h1`selbst benutzen und dessen Inhalt in der Größe so setzen, dass er genau die Maße vom Hintergrundbild hat.

GIF vs. PNG

Das Bild in Abbildung 10.5 ist ein GIF (genauer gesagt ein GIF89a) mit transparenten Bereichen. Eine visuell viel attraktivere Wahl wäre gewesen, ein PNG mit einem kompletten 8-bit-Alpha-Kanal zu verwenden, aber der IE/Win unterstützt Alpha-Kanäle bei PNG nicht korrekt. Wir werden das Warum und Weshalb für das Überwinden dieser Einschränkung am Ende des Projekts besprechen.

```
#pageHeader {background: url(pageHeader.jpg) 0 0 no-repeat;
  height: 157px; width: auto;}
#pageHeader h1 {background: url(ph-flower.gif) 0 0 no-repeat;
  height: 330px; width: 250px;}
</style>
```

Großartig! Außer dass es nun größer ist als sein Elternelement (`div#pageHeader`) und auf der linken Seite steht – kaum da, wo es eigentlich sein sollte. Also werden wir das `h1` absolut positionieren, damit es in seinem Container-Block ganz oben rechts sitzt.

Das große Z

Den Wert `101` für `z-index` haben wir mehr oder weniger zufällig gewählt. Jede Ganzzahl kann als z-Index-Wert benutzt werden. Die einzige Einschränkung besteht darin, dass höhere Zahlen *vor* niedrigeren Zahlen stehen müssen, und damit müssen alle positionierten Elemente, bei denen wir sicherstellen wollen, dass sie oben auf dem `h1` sitzen, `z-index`-Werte von `102` oder höher haben. Beachten Sie, dass `z-index` nur mit positionierten Elementen funktioniert.

```
#pageHeader h1 {background: url(ph-flower.gif) 0 0 no-repeat;
  height: 330px; width: 250px; position: absolute; z-index: 101;
  top: 0; right: 0; margin: 0;}
```

Damit wird das `h1` in der oberen rechten Ecke des `container` platziert, den wir am Ende des letzten Abschnitts in einen Container-Block verwandelt haben. Ein zusätzlicher Bonus ist, dass er durch die absolute Positionierung komplett aus dem normalen Fluss herausgenommen wird; also wirkt sich seine Höhe nicht mehr auf die Höhe des `div`-Elements `pageHeader` aus.

Ein Problem haben wir aber: Es reicht nicht, das `h1` einfach oben rechts in die Ecke des Containers zu stecken. Dadurch wird es sich noch lange nicht mit dem Rest des Mastheads ausrichten. Wenn wir noch einmal das Original-Design anschauen, sehen wir, dass sich die Spitze der Blume 95 Pixel über dem Top des Mastheads befindet und ihre rechte Kante 80 Pixel rechts von der rechten Kante der Haupttextspalte ist. Also legen wir diese Werte zugrunde und verschieben alles entsprechend an die richtige Stelle.

```
#pageHeader h1 {background: url(ph-flower.gif) 0 0 no-repeat;
  height: 330px; width: 250px; position: absolute; z-index: 101;
  top: -95px; right: -80px; margin: 0;}
```

Jetzt ist es dort, wo es hinsoll. Wir sollten nun nur noch dafür sorgen, dass der Text im `h1` nicht mehr angezeigt wird, damit wir das Ergebnis bekommen, das in Abbildung 10.6 zu sehen ist.

```
#pageHeader h1 {background: url(ph-flower.gif) 0 0 no-repeat;
  height: 330px; width: 250px; position: absolute; z-index: 101;
  top: -95px; right: -80px; margin: 0;}
#pageHeader h1 span {visibility: hidden;}
</style>
```

Obwohl beim Original-Design die Spitze der Blume nicht durch das Browser-Fenster abgeschnitten wurde, ist das ein netter Effekt, also behalten wir ihn bei. Falls wir es uns später einmal anders überlegen sollten, brauchen wir nur den oberen Rand des Containers auf etwas um `100px` einzustellen.

Abbildung 10.6

Über das `h1` haben wir die Blume an der gewünschten Stelle platziert.

Jetzt geht's an das `h2`, das bisher als einfacher schwarzer Text in der linken oberen Ecke des Mastheads erscheint. Er soll in dem hellgrünen Streifen platziert werden, also müssen wir ihn offensichtlich positionieren. Wir messen mal eben nach und finden heraus, dass seine Oberkante 134 Pixel unter dem Beginn des Mastheads sein sollte und seine rechte Kante 140 Pixel links davon. Also dann:

```
#pageHeader h1 span {visibility: hidden;}
#pageHeader h2 {position: absolute; z-index: 102;
  top: 134px; right: 140px; margin: 0; padding: 0;}
</style>
```

Natürlich sieht der Text absolut noch nicht so aus wie in dem Design aus Abbildung 10.1, also werden wir den Serifen-Text erst einmal etwas vergrößern und ihm Kursiv- und Fettschrift zuweisen. Und rechtsbündig machen wir ihn auch.

```
#pageHeader h2 {position: absolute; z-index: 102;
  top: 134px; right: 140px; margin: 0; padding: 0;
  font: bold italic 1.1em/1em Times, serif; text-align: right;}
```

Um die gewünschte Farbe zu bekommen, sampeln wir die Pixel im Original-Design. Wo wir gerade dabei sind – beachten Sie, dass alles außer dem Akronym »CSS« klein geschrieben ist, und die Buchstaben eine etwas größere Laufweite als normal haben.

```
#pageHeader h2 {position: absolute; z-index: 102;
  top: 134px; right: 140px; margin: 0; padding: 0;
  color: rgb(91,131,104);
  text-transform: lowercase; letter-spacing: 0.2em;
  font: bold italic 1.1em/1em Times, serif; text-align: right;}
```

Natürlich bekommt so auch »CSS« Kleinbuchstaben, also müssen wir darauf aufpassen, dass es Großbuchstaben bleiben. Das können wir dadurch erreichen, dass wir den Wert für `text-transform` überschreiben, den das Element `acronym` erbt, so wie

Beschnittene Fonts

In unseren Testläufen haben wir herausgefunden, dass einige Versionen des IE/Win im `h2` oben und unten die Buchstaben kappen, wenn das `h2` positioniert ist. Bei anderen Versionen ist das nicht der Fall. Wenn Ihnen dieses Problem unterkommt, versuchen Sie etwas wie `line-height: 1.2em; margin-top: -0.2em;`. Das sollte den Text an die richtige Stelle setzen und die »Verstümmelung« verhindern, weil sie scheinbar durch eine `line-height` von `1em` verursacht wird.

wir auch das geerbte `text-align` beim Container überschrieben haben. Das Ergebnis können Sie in Abbildung 10.7 sehen.

```
#pageHeader h2 {position: absolute; z-index: 102;
  top: 134px; right: 140px; margin: 0; padding: 0;
  color: rgb(91,131,104);
  text-transform: lowercase; letter-spacing: 0.2em;
  font: bold italic 1.1em/1em Times, serif; text-align: right;}
#pageHeader h2 acronym {text-transform: uppercase;}
</style>
```

Abbildung 10.7

Die Schönheit der Gestaltung mit CSS

Echter Text vs. Bild-Text

Die Verwendung des tatsächlichen Textes im `h2`-Element wirft eine interessante Frage auf: Wann ist es eine gute Idee, den Text zu stylen, und wann ist es besser, ihn durch ein Bild zu ersetzen? Dies ist, wie Sie sicher schon vermutet haben, ein kontroverses Thema. Einige meinen, dass man einen Text nie durch ein Bild ersetzen sollte, andere finden, je mehr in Bilder umgewandelter Text im Dokument steht, desto besser.

Die Begründung für das Ersetzen von Text durch ein Bild besteht üblicherweise darin, dass der »Bild-Text« normalerweise für die meisten Leute, die die Site besuchen, viel besser aussieht. Text in Bildern kann geglättet werden, man kann die Laufweite ändern, ihn anpassen und auf vielerlei andere Arten absolut fabelhaft aussehen lassen. Damit bekommt man eine viel größere visuelle Flexibilität als bei echtem Text, aber auf Kosten eines größeren Seitengewichts und einiger Zugänglichkeitsprobleme.

Demgegenüber kann richtiger Text gut aussehen oder auch nicht, und das hängt vom Betriebssystem des Users ab. Moderne Systeme wie Windows XP und Mac OS X verfügen über eine eingebaute Textglättung, über die eine Menge erreicht werden kann, wenn echter Text so gut aussehen soll wie Text in Bildern. Älteren Betriebssystemen (und Browsern) steht das nicht zur Verfügung. Echter Text hat den eindeutigen Vorteil, dass er das gesamte Seitengewicht deutlich weniger belastet und in höchstem Maße zugänglich ist.

Wir haben uns dafür entschieden, das `h2` *in diesem Projekt als echten Text zu belassen, aber es ist natürlich genauso einfach, ihn wie beim* `h1` *durch ein Hintergrundbild zu ersetzen. Das bedeutet größeres Seitengewicht, ist aber ein angemessener Preis dafür zu wissen, dass der Text wirklich schön aussehen wird. Am Ende müssen Sie hier (wie bei so vielen anderen Dingen) Ihr eigenes Urteilsvermögen bemühen, um eine Wahl zu treffen.*

10.4.4 Styling der Zusammenfassung

Wie Sie wahrscheinlich schon erraten haben, ist dies ein weiterer Bereich im Design, der ein Hintergrundbild nötig hat. Dieses Bild (die Maße sind 647 Pixel breit und 100 Pixel hoch) sehen Sie in Abbildung 10.8.

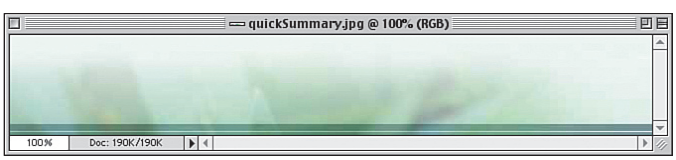


ABBILDUNG 10.8

Das Hintergrundbild für den Bereich der Zusammenfassung

Wie bei der Kopfzeile werden wir es in den Hintergrund der Zusammenfassung setzen, stellen aber keine explizite Höhe ein. Font-Styles und Hintergrund werden wir gleichzeitig setzen.

```
#pageHeader h2 acronym {text-transform: uppercase;}
#quickSummary {font: italic 1em/2 Times, "Times New Roman", serif;
  background: url(quickSummary.jpg) 0 100% no-repeat;}
</style>
```

Zusätzlich zur doppelten Zeilenhöhe (also dem Wert `2` für `line-height`) haben wir das Hintergrundbild in die linke untere Ecke des Hintergrunds dieses Elements gestellt. Wir werden die gleiche Farbe wie die der oberen Bildkante angeben, damit die Teile vom Hintergrund abgedeckt sind, die das Bild vielleicht nicht »erwischt«. Da wir gerade dabei sind, stellen wir auch die Textfarbe auf ein dunkles Grün.

```
#quickSummary {font: italic 1em/2 Times, "Times New Roman", serif;
  background: rgb(94%,98%,96%) url(quickSummary.jpg) 0 100% no-repeat;
  color: rgb(42,92,42);}
```

Hintergründe und Rahmen

Vielleicht ist Ihr erster Impuls, einen Rahmen zu benutzen, um die Separatoren zwischen der Zusammenfassung und den sie umgebenden Inhalt zu erstellen, und eine schlechte Idee ist das nicht. Wenn wir es mit einem gemusterten Hintergrund statt mit einem reinweißen Hintergrund zu tun gehabt hätten, dann wäre das auch unentbehrlich gewesen. In diesem Fall war es leichter, einfach Teile des Layouts auseinander zu schieben und dann dem Hintergrund die visuelle Arbeit zu überlassen. Dies ist in vielerlei Hinsicht eleganter, als wenn es über Rahmen erledigt wird.

Es wird Zeit, die Ränder und das Padding hinzuzufügen. Welche Ränder? Nun, schauen Sie sich noch einmal die Abbildung 10.1 an und achten Sie auf die dünnen weißen Linien entlang des oberen und unteren Bereichs der Zusammenfassung. Die werden wir nachbauen, indem wir dem Bereich der Zusammenfassung einen oberen und unteren Rand von einem Pixel geben. Dadurch »scheint« der weiße Hintergrund der Seite durch, und somit wird die Zusammenfassung vom Masthead davor und dem Hauptinhalt danach getrennt.

```
#quickSummary {font: italic 1em/2 Times, "Times New Roman", serif;
  background: rgb(94%,98%,96%) url(quickSummary.jpg) 0 100% no-repeat;
  color: rgb(42,92,42);
  margin: 1px 0;}
```

Was das Padding angeht – dafür ist ein interessanter Mix nötig! Um den Text in der Zusammenfassung von den Kanten seiner Element-Box fernzuhalten, werden wir oben, unten und links ein Padding in der Größenordnung von 1 - 1,5 em einfügen. Auf der rechten Seite brauchen wir allerdings genug Padding, damit Text und Blüte sich nicht überlappen. Weil die Blüte ein Bild ist und darum eine auf Pixeln basierende Breite aufweist, wird das Padding der Zusammenfassung auf dieser Seite in Pixel bemessen.

```
#quickSummary {font: italic 1em/2 Times, "Times New Roman", serif;
  background: rgb(94%,98%,96%) url(quickSummary.jpg) 0 100% no-repeat;
  color: rgb(42,92,42);
  margin: 1px 0; padding: 1em 180px 1.5em 1.5em;}
```

»Moment mal«, könnten Sie jetzt sagen. »Der Text hätte doch auch über die Absatzränder von den Kanten der Zusammenfassung ferngehalten werden können.« Das stimmt zum ersten nicht, weil die Absatzränder aus der Zusammenfassung herausgeragt und die Zusammenfassung dann visuell vom Masthead und dem Hauptinhalt weggeschoben hätten. Zweitens werden die Absätze keine Ränder bekommen, die sich auf das Layout auswirken.

Achten Sie darauf, dass im Original-Design die Zusammenfassung wie ein einzelner Textabsatz erscheint, obwohl er zwei `p`-Elemente enthält. Um diesen Effekt zu reproduzieren, lassen wir diese Absätze Inline-Boxen generieren, und das Ergebnis sehen Sie in Abbildung 10.9.

```
#quickSummary {font: italic 1em/2 Times, "Times New Roman", serif;
  background: rgb(94%,98%,96%) url(quickSummary.jpg) 0 100% no-repeat;
  color: rgb(42,92,42);
  margin: 1px 0; padding: 1em 180px 1.5em 1.5em;}
#quickSummary p {display: inline;}
</style>
```

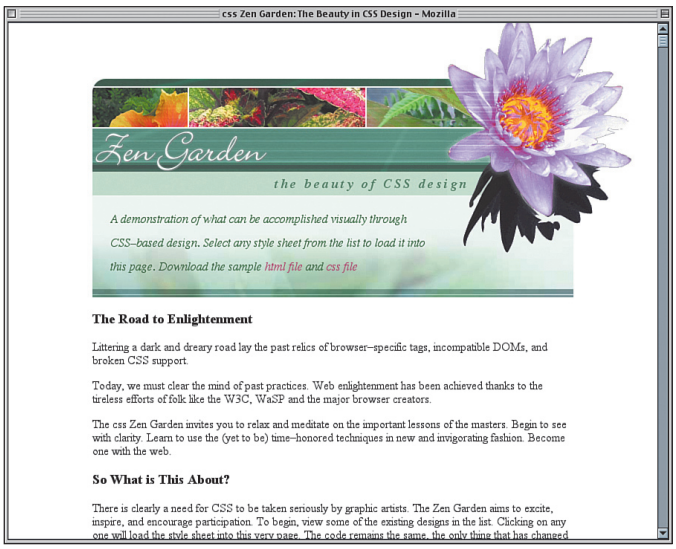


Abbildung 10.9

Der Bereich der Zusammenfassung in seiner ganzen gestylten Schönheit

Wenn Sie hier den IE/Win benutzen, fällt Ihnen sicherlich auf, dass die beiden Absätze nun dicht aneinander gezwängt sind, als gäbe es keinen Platz zwischen ihnen. Natürlich ist da Platz, aber das braucht Sie jetzt nicht zu kümmern; wir werden sie auseinander ziehen, indem wir beim zweiten Absatz einen linken Rand einfügen. Wir können das machen, denn seine Klasse ist als `p2` gesetzt.

```
#quickSummary p {display: inline;}
#quickSummary p.p2 {margin-left: 0.25em;}
</style>
```

Damit bekommt die ersten Zeile im Element auf der linken Seite ein Viertel em Rand, anders gesagt: genau am Anfang des zweiten Absatzes. Damit wird bei allen anderen Browsern außer dem Internet Explorer eine etwas größere Trennung als normal erzielt, aber nicht so viel, dass es schlecht aussähe.

Wir schreiben bereits Styles, die speziell für den zweiten Absatz sind, also fügen wir nun am Ende des Satzes noch einen Punkt ein. Das HTML dürfen wir zwar nicht ändern, aber wir können etwas generierten Inhalt einfügen.

```
#quickSummary p.p2 {margin-left: 0.25em;}
#quickSummary p.p2:after {content: ".";}
</style>
```

Nein, der Explorer wird den generierten Inhalt nicht darstellen, aber das ist okay. Hier geht es nur um einen kleinen Effekt, und den Nutzern des Internet Explorers wird nichts entgehen.

10.4.5 Styling des Hauptinhalts

Jetzt, wo der obere Bereich des Designs vollständig ist, wollen wir unsere Aufmerksamkeit auf den Hauptinhalt des Dokuments richten. Hier wird es nun richtig interessant, weil wir sehr kreativ werden müssen, um das uns vorgegebene visuelle Design zu reproduzieren. Ein Vergleich des Ausschnitts aus der Dokumentstruktur in Listing 10.3 mit dem in Abbildung 10.1 gezeigten Design zeigt, warum.

Doppelt hält besser

Listing 10.3 ist ein Ausschnitt aus Listing 10.1 und wird hier erneut gezeigt, um auf die »Content«-Teile des Dokuments aufmerksam zu machen, ohne dass der Rest der Dokumentstruktur davon ablenkt.

Listing 10.3 Ein Teil der Dokumentstruktur

```
<div id="intro">
    [...Kopfzeile und Überblick...]
    <div id="preamble">
      <h3><span>The Road to Enlightenment</span></h3>
      [...Inhalt...]
    </div>
  </div>
  <div id="supportingText">
    <div id="explanation">
      <h3><span>So What is This About?</span></h3>
      [...Inhalt...]
    </div>
    <div id="participation">
      <h3><span>Participation</span></h3>
      [...Inhalt...]
    </div>
    <div id="benefits">
      <h3><span>Benefits</span></h3>
      [...Inhalt...]
    </div>
    <div id="requirements">
      <h3><span>Requirements</span></h3>
      [...Inhalt...]
    </div>
    <div id="footer">
      [...Inhalt...]
    </div>
  </div>
```

Da sehen Sie beispielsweise einen dünnen grünen Rahmen auf der linken Seite des Hauptinhalts, der sich von der Zusammenfassung nach unten bis zum Footer erstreckt. (Schauen Sie noch einmal die Abbildung 10.1 an, wenn Sie sich nicht erinnern.) Normalerweise müssten wir jetzt nur einfach bei dem `div`-Element, in dem der gesamte Text steht, einen linken Rahmen einfügen. Dieses Dokument verfügt allerdings nur über ein `div`-Element, auf das dies zutrifft – das mit dem Individualformat (`id`) `supportingText`. Der Footer befindet sich ebenfalls in diesem `div`-Element, also würde sich jeder Rahmen, den wir an seiner Seite einfügen, bis unter den Footer ausdehnen. Obendrein ist die Präambel (der erste Absatz in der Spalte mit dem Haupt-

inhalt) außerhalb von `supportingText`, also würde der Rahmen gar nicht daneben erscheinen. Beides ist inakzeptabel.

Ein anderes Beispiel: Die Liste der Links, die in der Seitenleiste auf der rechten Seite erscheint, hat einen Hintergrund, der sich über die gesamte Höhe des Inhalts erstreckt. Weil die Links nach dem Footer kommen, müssen wir sie an Ort und Stelle positionieren. Positionierte Elemente sind allerdings nur so groß wie ihr Inhalt. Wenn wir wollen, dass diese Seitenleiste sich visuell von der Zusammenfassung bis zum Footer erstreckt, dann müssen wir das anders bewerkstelligen, als einfach nur der Linkliste einen Hintergrund zu geben.

Um dieses letzte Problem werden wir uns als Erstes kümmern. Wir wissen, dass wir die Linkliste positionieren werden, also müssen wir die Spalte mit dem Hauptinhalt beiseite räumen. Das könnte auch über einen rechten Rand klappen, aber stattdessen werden wir bei der Präambel und dem darauf folgenden Text ein einheitliches Padding rechts anwenden. Den genauen Wert für das Padding haben wir durch die Abmessung der Breite des Seitenleistenhintergrunds und des Rahmens, der ihn von der Inhaltsspalte abtrennt, ermittelt.

```
#quickSummary p.p2:after {content: ".";}
#preamble, #supportingText {padding-right: 217px;}
</style>
```

Das Padding verschafft uns Platz, damit wir bei dem Hintergrund der Seitenleiste anfangen und gleichzeitig den Farbverlauf im Balken an der rechten Kante des Inhalts einführen können – siehe Abbildung 10.10.

```
#preamble, #supportingText {padding-right: 217px;}
#preamble {background: url(side.jpg) 100% 100% repeat-y;}
</style>
```

Abbildung 10.10

Der erste Teil der Seitenleiste und der Hintergrund für den Hauptinhalt wird eingefügt.

In Abbildung 10.10 sehen Sie ein Bild für den Hintergrund mit der Größe 302 Pixel breit und 4 Pixel hoch, das in der rechten unteren Ecke der Präambel beginnt und sich dann vertikal wiederholt – sowohl nach oben als auch nach unten, obwohl keine Wiederholung nach unten sichtbar ist, denn das Bild fängt unten beim Element an.

Dieses Bild enthält nicht nur das Hintergrundmuster der Seitenleiste, sondern auch die vertikale Linie, die zwischen der Seitenleiste und dem Hauptinhalt verläuft, und auch den Balken mit dem Verlauf von rechts nach links, der nun hinter dem Hauptinhalt erscheint. Dies alles ist nur ein einziges Bild mit vertikalen Wiederholungen. Weil das rechte Padding der Präambel gleich dem Teilstück Seitenleiste plus Separator des Hintergrundbildes ist, kann der Inhalt über dem Farbverlauf im Balkenbereich des Hintergrunds platziert werden.

Damit sich das Muster wiederholt, brauchen wir nur das gleiche Bild dem Hintergrund des darüber liegenden »supportingText«-`div`s hinzuzufügen und es in der rechten oberen Ecke beginnen zu lassen und vertikal zu wiederholen.

```
#preamble {background: url(side.jpg) 100% 100% repeat-y;}
#supportingText {background: url(side.jpg) 100% 0 repeat-y;}
</style>
```

Erinnern Sie sich noch an den Rahmen auf der linken Seite des Designs? Den wollen wir jetzt passend einfügen. Wie schon besprochen, können wir `div#supportingText` nicht verwenden, weil der Footer darin enthalten ist, aber nicht die Präambel. Wir können allerdings bei jedem der »preamble«-, »explanation«-, »participation criteria«-, »benefits«- und »requirements«-`div`s einen Rahmen einfügen. Eine gute Sache, dass alle `div`s Individualformate haben, die wir benutzen können! Wie es aussieht, wenn wir allen diesen `div`-Elementen Rahmen geben, können Sie in Abbildung 10.11 sehen.

ABBILDUNG 10.11

Eine Reihe von `div`s bekommen Rahmen.

```
#supportingText {background: url(side.jpg) 100% 0 repeat-y;}
#preamble, #explanation, #participation, #benefits, #requirements {
  border-left: 1px solid rgb(184,214,194);}
</style>
```

Aber Moment mal! Der Rahmen ist zerstückelt, und in der Seitenleiste ist eine Riesenlücke. Wie konnte das passieren?

Das ist passiert, weil die Ränder der `h3`- und `p`-Elemente auf die – in diesem Fall *durch* – die Ränder der `div`-Elemente zusammengefallen sind. Das ist ein Teil der Art und Weise, wie ein Zusammenfallen der Ränder funktioniert. Damit wird es möglich, dass die Ränder eines Elements aus der Element-Box seines Eltern-Elements herausragen.

Zum Glück gibt es eine Möglichkeit, dies zu umgehen. Wenn die `div`-Elemente über oberes und unteres Padding verfügen, werden sie sich ausdehnen, um ihre Element-Nachkömmlinge und deren gesamte Ränder aufzunehmen. Also werden wir einfach jedem dieser `div`-Elemente ein Pixel Padding oben und unten geben.

```
#preamble, #explanation, #participation, #benefits, #requirements {
  border-left: 1px solid rgb(184,214,194);
  padding-top: 1px; padding-bottom: 1px;}
```

Damit werden die Rahmen zusammentreffen, wie in der nächsten Abbildung zu sehen ist. Vorher wollen wir aber erst noch den eigentlichen Inhalt dieser `div`-Elemente stylen. Die Absätze sollen beispielsweise nicht gegen die Kanten der Hauptspalte laufen, also tut ihnen auch ein wenig Padding gut. Wir sollten ebenfalls die Trennung zwischen den Absätzen über Ränder definieren, die Textgröße etwas herunterschrauben und die Textzeilen mit einer aufgestockten `line-height` auseinanderziehen.

```
#preamble, #supportingText {padding-right: 217px;}
#preamble p, #supportingText p {font-size: 90%; line-height: 1.66em;
   margin: 0 1.5em; padding: 0.5em 0;}
#preamble {background: url(side.jpg) 100% 100% repeat-y;}
```

Auf ähnliche Weise können wir den `h3`-Elementen dieser `div`-Elemente passende Ränder, Farben und Text-Styles geben.

```
#preamble p, #supportingText p {font-size: 90%; line-height: 1.66em;
   margin: 0 1.5em; padding: 0.5em 0;}
#preamble h3, #supportingText h3 {letter-spacing: 0.1em;
  font: italic 1.2em Times, "Times New Roman", serif;
  color: rgb(107,153,139); margin: 1em 0 0.5em 0.5em;}
#preamble {background: url(side.jpg) 100% 100% repeat-y;}
```

Der obere Bereich des Hauptinhalts könnte noch eine besondere Note vertragen. Wenn Sie sich das Original-Design genau anschauen, sehen Sie einen grünen Rahmen, der oben zwischen Hauptinhalt und Seitenleiste verläuft und die gleiche Farbe hat wie die Rahmen links, die wir gerade eingefügt haben. Wir können diesen Rahmen passend unterbringen, wenn wir der Präambel einen oberen Rahmen geben – denken Sie daran, dass er sich wirklich von einer Seite des Containers bis zur anderen erstreckt, wie man in Abbildung 10.12 sehen kann.

Zusammenfallende Ränder

Es mag absurd erscheinen, dass die Ränder eines Elements aus dessen Eltern-Element herausragen dürfen, aber es ist notwendig, um gewöhnliches Layout zu schaffen. Stellen Sie sich eine Situation vor, bei der das erste Element in einem `div`-Element eine unsortierte Liste ist. Nehmen wir an, dass der obere Rand bei allen `div`-Elemente jeweils 1 em, bei allen unsortierten Listen 1,5 und den Listenelementen 0,5 em ist. Dank des Zusammenfallens der Ränder wird der Gesamtbetrag für die Ränder vor dem ersten Listenelement 1,5 em betragen. Gäbe es kein Zusammenfallen der Ränder, dann wäre vor diesem ersten Listenelement 3 em Platz.

Creeping-Text-Bug

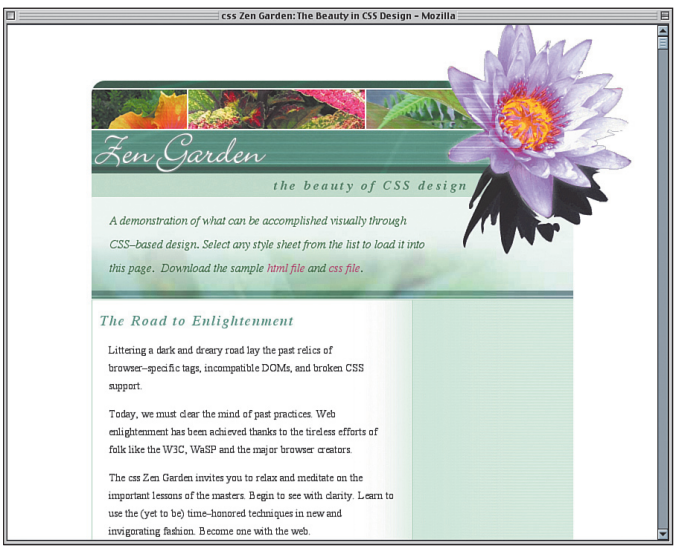
Den Usern des IE/Win ist wohl aufgefallen, dass der Haupttext sich plötzlich nach links verschiebt und nahe am Ende des Dokuments sogar abgeschnitten wird. Das wird weitgehend behoben sein, wenn wir mit dem Projekt fertig sind, aber Sie können mehr über diesen Bug und wie man ihn vermeidet unter `http://www.positioniseverything.net/explorer/creep.html` lesen.

```
#preamble {border-top: 1px solid rgb(184,214,194);
  background: url(side.jpg) 100% 100% repeat-y;}
```

Mittlerweile haben wir nun den Hauptinhalt weitgehend fertig gestylt, aber eine Sache fehlt noch, und das ist der blumige Hintergrund im unteren Bereich der Spalte.

ABBILDUNG 10.12

Die Rahmen treffen aufeinander, und der Haupttext wirkt besser gestylt.



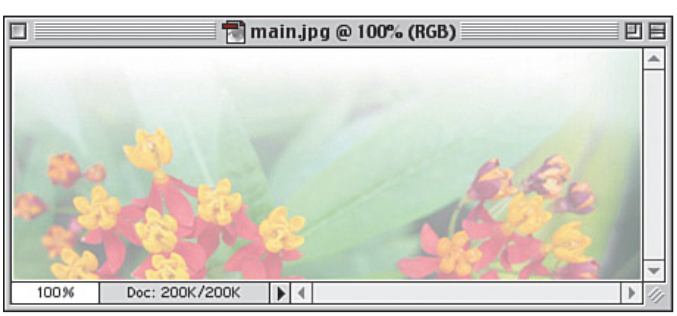
10.4.6 Dekorativer Footer

In der visuellen Original-Datei gab es im unteren Spaltenbereich des Hauptinhalts einen verwaschenen Hintergrund, also brauchen wir einen solchen auch in unseren Styles. Der Trick besteht darin, dafür das richtige Element auszuwählen, und in diesem Fall ist es das `div`-Element mit dem Individualformat `requirements`. Das `div`-Element `supportingText` kommt dafür nicht in Frage, weil es bereits einen Hintergrund hat und obendrein auch über den Footer hinausgeht. Also werden wir das `requirements-div` nehmen.

Das dazugehörige Bild sehen Sie in Abbildung 10.13; es ist 429 Pixel breit und 159 Pixel hoch.

ABBILDUNG 10.13

Das Bild für den Bereich der unteren Spalte des Hauptinhalts



Beachten Sie den Farbverlauf von rechts nach links in der rechten oberen Ecke des Bildes. Wenn wir unsere Styles richtig hinkriegen, wird sich das an den bereits vorhandenen Farbverlauf anschließen und zu einem nahtlosen Übergang zwischen beiden führen.

Achten Sie auch auf den weißen Streifen an den Seiten und unten im Bild. Darüber erreichen wir eine visuelle Trennung zwischen dem Hintergrund und den Rahmen, die ihn umgeben. Der naheliegende erste Schritt ist, das Bild in den Hintergrund zu setzen; wo wir gerade dabei sind, geben wir dem Element auch gleich einen unteren Rahmen.

```
#preamble, #explanation, #participation, #benefits, #requirements {
  border-left: 1px solid rgb(184,214,194);
  padding-top: 1px; padding-bottom: 1px;}
#requirements {border-bottom: 1px solid rgb(184,214,194);
  background: url(main.jpg) 100% 100% no-repeat;}
</style>
```

Warum das Bild in die untere rechte Ecke kommt? Um ein kleines Problem im IE5/Mac zu vermeiden, der die Platzierung im Hintergrund mit Bezug auf die äußere Rahmenkante des Elements statt der äußeren Kante des Paddings berechnet (anderweitig auch als innere Rahmenkante bezeichnet). Wenn wir es in die untere linke Ecke gestellt hätten, wäre der weiße Streifen an der linken Seite des Hintergrundbildes unter den linken Rahmen gerutscht. Da jedoch die rechte Seite des Elements keinen Rahmen hat, befinden sich Rahmen- und Paddingkante an der gleichen Stelle, und das Layout ist einheitlicher.

Ein kurzer Kontrollblick auf das erzielte Design zeigt, dass der Text ein wenig zu weit in den Hintergrund dringt, den wir gerade eingefügt haben, und das macht die letzten paar Zeilen schwerer lesbar. Das können wir leicht korrigieren: Wir werden einfach dem `div`-Element ein unteres Padding hinzufügen, das Ergebnis sehen wir in Abbildung 10.14.

```
#requirements {border-bottom: 1px solid rgb(184,214,194);
  background: url(main.jpg) 100% 100% no-repeat;
  padding-bottom: 100px;}
```

Jetzt können wir unsere Aufmerksamkeit dem eigentlichen Footer zuwenden – genauer gesagt dem `div`-Element mit dem Individualformat `footer`. Darin sind nur wenige Inhalte: fünf Hyperlinks. Mehr nicht! Mit einem `div`-Element und fünf Links müssen wir es irgendwie hinkriegen, dass es wie der Footer im Design aussieht.

Die erste Herausforderung besteht darin, ihn überhaupt auf die richtige Breite zu kriegen. Wenn Sie sich erinnern – der Footer ist im `div` mit der `id supportingText` enthalten, und dieses `div` hat ein Padding von `217px`. Wie wir in Abbildung 10.15 sehen können, ist das Bild, das wir in den Hintergrund des Footers stellen müssen, etwas breiter. Genau genommen ist es 647 Pixel breit und 123 Pixel hoch.

Ein kleiner Unterschied

Ein Unterschied zwischen IE5/Mac und anderen Browsern ist immer noch sichtbar. Der weiße Streifen an der Unterkante rutscht unter den unteren Rahmen des `div`-Elements. Ein Workaround dafür wäre, den unteren Rahmen des Elements zu entfernen und ihn im Hintergrundbild mit einem Streifen unten zu simulieren. Das machen wir aber nicht, sondern akzeptieren einfach diesen kleinen Unterschied im IE5/Mac und fahren fort.

Abbildung 10.14

Die Spalte des Hauptinhalts wird durch den Hintergrund nun abgeschlossen.

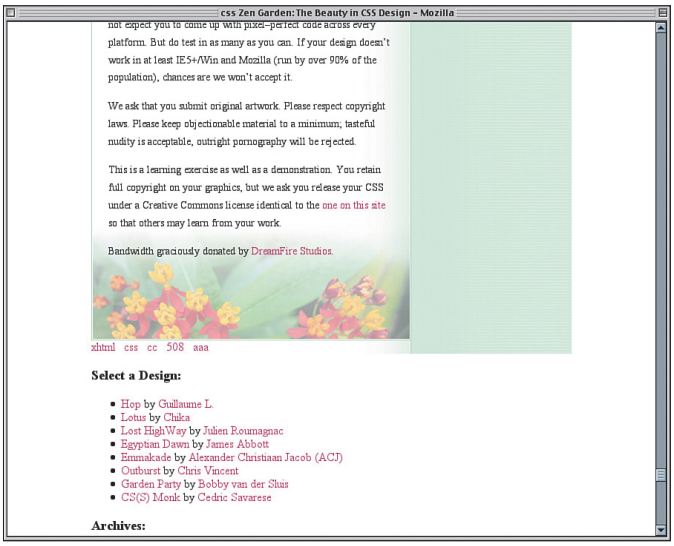
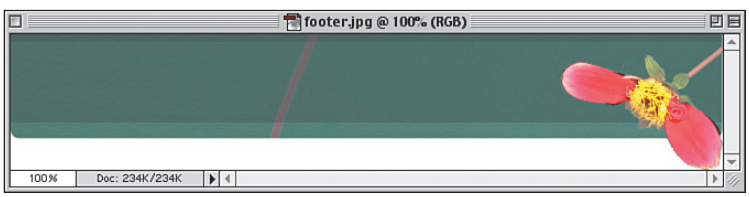


Abbildung 10.15

Das Bild, das wir hinter den Inhalt des Footers setzen wollen



Dank des Paddings ist der Bereich, der im »supportingText«-`div` enthalten ist, genau 430 Pixel breit (647 - 217). Also haben wir einen 647 Pixel breiten Footer, können aber nur mit 430 Pixeln arbeiten. Wir brauchen nur aus diesem Inhaltsbereich auszubrechen und gleichzeitig die Höhe des Footers einzustellen.

```
#requirements {border-bottom: 1px solid rgb(184,214,194);
  background: url(main.jpg) 100% 100% no-repeat;
  padding-bottom: 100px;}
#footer {margin: 0 -217px 0 0; height: 123px;}
</style>
```

Indem wir dem Footer einen negativen rechten Rand geben, der genau gleich dem rechten Padding seines Elternelements ist, haben wir im Grunde visuell gesehen das Padding wirkungslos gemacht. Der Inhaltsbereich des Footers selbst ist nun 647 Pixel breit, und das lässt uns ganz genau den Raum, den wir für die Darstellung des Bildes brauchen. Also setzen wir es nun ein!

```
#footer {margin: 0 -217px 0 0; height: 123px;
  background: #FFF url(footer.jpg) 100% 1px no-repeat;}
```

Warum haben wir es wieder nach rechts gepackt? Das liegt nicht an dem IE5/Mac, wie sich herausstellt. Wir machen es, um einen Bug im IE6/Win zu vermeiden.

Der Bug ist deswegen recht merkwürdig, weil er scheinbar der linken Seite des Footers einen negativen Rand von einigen Pixeln hinzufügt. Es gibt keinen Grund auf Erden, warum er das machen sollte, aber er tut es einfach. Als Folge wird der Hintergrund des Footers nach links verschoben und auch abgeschnitten. Das wirft die Anordnung des Designs über den Haufen und sieht außerdem ziemlich schrecklich aus. Indem wir den Hintergrund des Footers nach rechts verlegen, können wir dem Bug ausweichen. Es ist egal, wie viel negativer Rand der IE/Win fälschlicherweise anwendet, wenn das Hintergrundbild dort niemals auftaucht.

Nun müssen wir bloß das Padding des Footers nutzen, um die Links an Ort und Stelle zu bringen. Über ein wenig Experimentieren finden wir heraus, dass wir etwa 60 Pixel oberes Padding brauchen, um den Text an die gewünschte Stelle nach unten zu bewegen. Wir fügen einfach nur so ein 1 em Padding rechts ein und ein halbes em Padding links, um den ersten Link davor zu bewahren, der linken Kante des Footers zu dicht auf die Pelle zu rücken.

```
#footer {margin: 0 -217px 0 0; height: 123px;
  background: #FFF url(footer.jpg) 100% 1px no-repeat;
  padding: 60px 1em 0 0.5em;}
```

Nun brauchen wir nur noch die eigentlichen Links zu ihrem Erscheinungsbild im visuellen Design passend zu stylen (jedenfalls so gut wie möglich). Wir müssen vor allem auf die Punkte Farbe und Größe achten, obwohl wir sie über `line-height` und `font-weight` ganz gut dem angleichen können, was ursprünglich designt worden ist (siehe Abbildung 10.16).

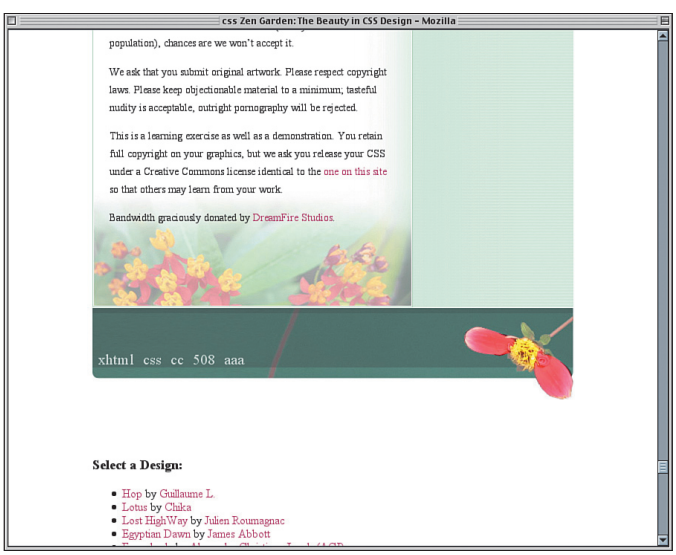


ABBILDUNG 10.16

Der vollständig gestylte Footer

```
#footer {margin: 0 -217px 0 0; height: 123px;
  background: #FFF url(footer.jpg) 100% 1px no-repeat;
  padding: 60px 1em 0 0.5em;}
#footer a {color: rgb(207,216,214); line-height: 1em;
  font-size: 1.25em; font-weight: 100;}
</style>
```

Damit haben wir also die Hauptspalte abgeschlossen! Nun brauchen wir nur noch die Seitenleiste zu platzieren und zu stylen, und wir sind fertig.

10.4.7 Styles für die Seitenleiste

Dies ist wirklich eine der leichtesten Übungen dieses Projekts, weil wir im Prinzip bloß die Liste mit den Links an den gewünschten Ort stellen und deren Inhalt stylen müssen. Dank der bisher geleisteten Arbeit brauchen wir uns nicht mehr um den Hintergrund oder andere Sachen zu kümmern. Das haben wir schon abgehakt.

Also positionieren wir nun die Link-Liste. Als Erstes positionieren wir sie in der oberen rechten Ecke ihres Container-Blocks und definieren ihre Breite auf 216 Pixel (die gleiche Breite wie bei der Seitenleiste).

```
#footer a {color: rgb(207,216,214); line-height: 1em;
  font-size: 1.25em; font-weight: 100;}
#linkList {position: absolute; z-index: 11;
  width: 216px; top: 0; right: 0;}
</style>
```

Eine zerbrechliche Lösung

Beachten Sie, dass diese Lösung nur funktioniert, weil der Text der Zusammenfassung drei Zeilen lang ist. Wenn der Text über zwei oder vier Zeilen ginge, wäre der Inhalt der Seitenleiste nicht an der korrekten Stelle. Wir werden das so lassen, hauptsächlich deswegen, weil es praktisch unmöglich zu umgehen ist, ohne in die Dokumentstruktur einzugreifen, denn das dürfen wir ja nicht. Wenn wir das dürften, könnten wir über einige einfache Änderungen der Struktur komplett verhindern, dass der Seitenleisteninhalt falsch ausgerichtet wird.

Wie immer ist der Container-Block eines absolut positionierten Elements das nächste Vorfahr-Element, das absolut oder relativ positioniert ist. Für die Link-Liste ist das der Container selbst, also wird die Blume momentan von der Link-Liste auf ganz unschöne Weise überlappt. Wir müssen die Liste in der Seitenleiste weiter nach unten schaffen, wo sie auch hingehört.

Das stellt uns eine gewisse Herausforderung, aber keine sonderlich große. Über das Ausmessen erfahren wir, dass die Entfernung von der oberen Kante des Containers bis zur Oberkante der Zusammenfassung 157 Pixel beträgt. Also kein Problem. Aber was ist mit der Höhe der Zusammenfassung? Die ist nicht in Pixel eingestellt, also hängt sie stattdessen von der Höhe ihres Inhalts ab. Wenn wir das Padding oben und unten und die Zeilenhöhen innerhalb der Zusammenfassung addieren, erhalten wir als Ergebnis 8,5 em. Dem geben wir noch etwas zu und erhalten so 8,6 em. Um die beiden addieren zu können, werden wir den `top` der Link-Liste auf `157px` einstellen und den `margin-top` auf `8.6em` – das Ergebnis sehen Sie in Abbildung 10.17.

```
#linkList {position: absolute; z-index: 11;
  width: 216px; top: 157px; right: 0;
  margin-top: 8.6em;}
```

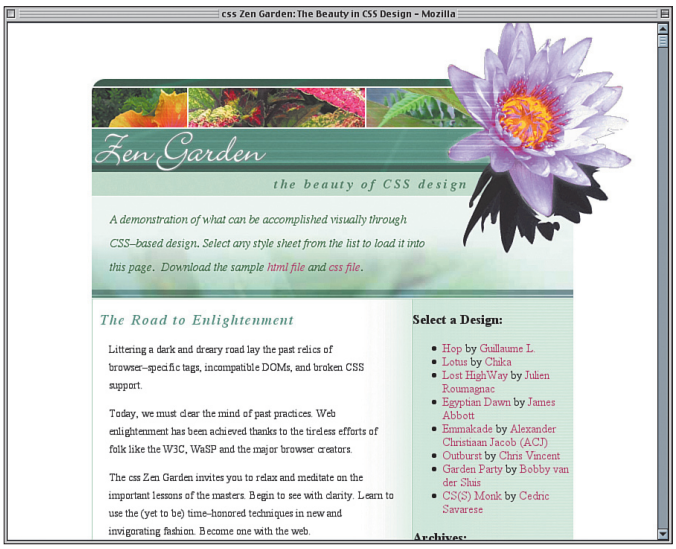


Abbildung 10.17

Die Link-Liste ist positioniert und kann gestylt werden.

Okay, die Links sind dort, wo wir sie haben wollen, aber sie ähneln dem Original-Design nicht sonderlich. Unsere Listen haben beispielsweise Bullets, die aber im Design nicht vorkommen. Also werden wir sie aus den Listenelementen entfernen.

```
#linkList {position: absolute; z-index: 11;
  width: 216px; top: 157px; right: 0;
  margin-top: 8.6em;}
#linkList li {list-style: none;}
</style>
```

Wir wollen die Einrückung der Listen nicht verlieren, weil sie eine Eigenschaft des Designs war, aber wir wollen die Einrückung auch nicht dem Zufall überlassen. Also werden wir explizit Ränder für die unsortierten Listen innerhalb des `div`s der Link-Listen einstellen und darauf achten, dass sie kein Padding haben.

```
#linkList {position: absolute; z-index: 11;
  width: 216px; top: 157px; right: 0;
  margin-top: 8.6em;}
#linkList ul {margin: 0.5em 1em 0 2em; padding: 0;}
#linkList li {list-style: none;}
```

Die Links selbst könnten auch etwas Unterstützung gebrauchen, denn sie sollen kursiv, grün und etwas größer als normaler Text werden. Und im Design sind sie auch alle in Kleinbuchstaben. Schnell ein paar Deklarationen eingestellt, und schon haben wir alles – siehe Abbildung 10.18.

```
#linkList li {list-style: none;}
#linkList a {color: rgb(99,131,101);
  font: italic 1.15em Times, serif;
  text-transform: lowercase;}
</style>
```

Abbildung 10.18

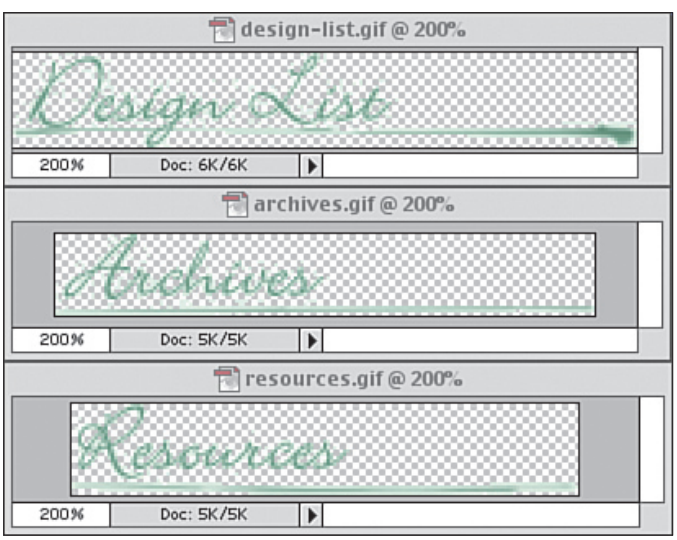
Die Listen innerhalb der Seitenleiste erhalten ein grundlegendes Styling.

Das ist ein Fortschritt, aber ganz plötzlich wird es in der Seitenleiste auch etwas eng. Wir müssen die Einträge auseinander ziehen, und dazu kommen wir auch gleich, aber zuerst schauen wir uns noch die drei `h3`s an (»Design List« usw.).

In Abbildung 10.1 haben sie alle einen schönen Skript-Font und besondere Unterstrich-Effekte. Das überschreitet offensichtlich weit die Möglichkeiten mit normalem Text, also kommen die drei Bilder aus der Abbildung 10.19 zum Einsatz.

Abbildung 10.19

Diese drei Bilder ersetzen den Überschriftentext in der Seitenleiste.



Wie Sie sehen können, handelt es sich bei den Bilder um GIFs mit transparenten Bereichen, wodurch sie gut mit dem Hintergrund der Seitenleiste verschmelzen können.

Um sie zu platzieren, werden wir nun alle drei `h3`-Elemente der Seitenleiste auf einmal stylen. Dabei werden wir nicht nur für alle Elemente gleiche Ränder setzen, sondern auch die gleiche Breite und Höhe. Wir geben ihnen allen auch das »Resources«-Bild als Hintergrund, unterbinden eine Wiederholung und platzieren es vertikal zentriert und 10 Pixel rechts neben die linke Kante des Elements.

```
#linkList a {color: rgb(99,131,101);
  font: italic 1.15em Times, serif;
  text-transform: lowercase;}
#linkList h3 {margin: 1em 0 0; width: 216px; height: 35px;
  background: url(resources.gif) 10px 50% no-repeat;}
</style>
```

Nachdem dies nun erledigt ist, können wir alles so stehen lassen und nur die Bilder ändern, die im Hintergrund der Überschriften für »Design List« und »Archives« verwendet werden.

```
#linkList h3 {margin: 1em 0 0; width: 216px; height: 35px;
  background: url(resources.gif) 10px 50% no-repeat;}
#lselect h3 {background-image: url(design-list.gif);}
#larchives h3 {background-image: url(archives.gif);}
</style>
```

Wenn wir diesen Ansatz wählen, wird es viel einfacher, die Styles der Überschriften als Gruppe zu überarbeiten. Falls wir später einmal beschließen, die Platzierung der Bilder innerhalb der Überschriften zu ändern, brauchen wir nur die `background-position`-Schlüsselworte in der Regel `#linkList h3` editieren.

Als Letztes müssen wir bei den Überschriften noch den Text aus dem Vordergrund entfernen, und das führt dann schließlich zum Endergebnis in Abbildung 10.20.

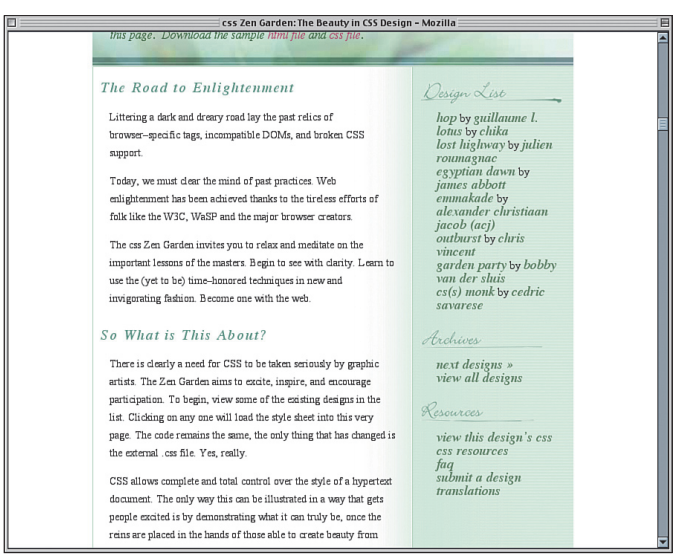


ABBILDUNG 10.20

Die Überschriften werden durch Bilder ersetzt.

```
#larchives h3 {background-image: url(archives.gif);}
#linkList h3 span {display: none;}
</style>
```

Nun geht es an die Links – die waren schon viel zu lange so eingepfercht. Weil die Links der »Design List« anders gestylt sind als die in den anderen beiden Abschnitten der Seitenleiste, werden wir sie uns der Reihe nach vornehmen.

Wenn Sie sich die Einträge in der »Design List« im visuellen Design genau anschauen, werden Sie sehen, dass bei jedem Eintrag die zweite Zeile (die Zeile mit dem Autoren) ein wenig kleiner ist als normal. Der Name des jeweiligen Designs ist allerdings so groß wie regulärer Text. Wenn wir nur die Listenelemente in dieser Liste auf etwa 85 % der Normalgröße einstellen, kommen wir diesem Effekt ziemlich nahe. Das liegt daran, dass wir bereits eine Regel (`#linkList a`) geschrieben haben, die die Links in der Seitenleiste auf das 1,15-fache der Größe der `font-size` ihrer Elternelemente setzen. Wenn wir die Größe der Listenelemente auf das 0,85-fache des Normalen reduzieren, werden die Links das 0,9775-fache der Normalgröße haben – beinahe 1,0, damit wir zufrieden sind.

Also machen wir das so und geben den Listenelementen gleichzeitig einen unteren Rand.

```
#linkList h3 span {display: none;}
#lselect li {font-size: 85%; margin-bottom: 1.5em;}
</style>
```

In Ordnung – nun also zu den Links selbst. Jeder Designname muss in seiner eigenen Zeile sitzen und in Fettschrift sein. Der Name soll in Kleinbuchstaben geschrieben und die Buchstaben etwas auseinander gezogen sein. Das machen wir ganz einfach für alle Links im Abschnitt »Design List«.

```
#lselect li {font-size: 85%; margin-bottom: 1.5em;}
#lselect li a {display: block; font-weight: bold;
  letter-spacing: 0.2em; text-transform: lowercase;}
</style>
```

Huch – wir haben diese Styles gerade auf alle Links in der »Design List« angewendet, einschließlich der Links mit den Autorennamen, die absolut nicht so aussehen sollen. Tatsächlich sollen sie nur in Fettschrift sein – in Größe und Aussehen wie das Wort »by«, das neben jedem Namen sitzt. Und was das angeht – die Namen der Autoren müssen natürlich neben dem Wort »by« stehen, was heißt, dass sie nicht Blocklevel sein können. Mit anderen Worten – wir müssen alle Styles rückgängig machen, die wir gerade auf sie angewendet haben.

Glücklicherweise haben die Links mit den Autorennamen eine Sache gemeinsam: die `class` mit dem Namen `c`. Nach dieser kann sich unser Selektor richten und wir können das Erscheinungsbild der Links in nur vier Deklarationen reparieren – schauen Sie sich Abbildung 10.21 an.

```
#lselect li a {display: block; font-weight: bold;
  letter-spacing: 0.2em; text-transform: lowercase;}
#lselect li a.c {display: inline;
  font: bold 1em Times, serif;
  letter-spacing: 0; text-transform: none;}
</style>
```

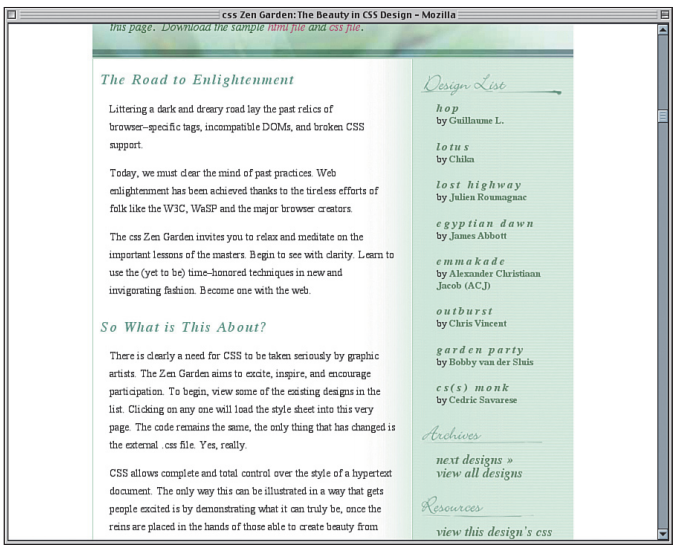


Abbildung 10.21

Die Links der »Design List« werden lockerer verteilt und wie gewünscht formatiert.

Wir stehen schon sehr, sehr dicht vor dem Finale. Wir brauchen in der Tat nur noch dafür zu sorgen, dass die Links in den Abschnitten »Archives« und »Resources« so wie im Original-Design ausschauen. Obwohl sie ein wenig auseinander gezogen sind, ist das doch weniger als bei den Links im Abschnitt »Design List«. Also sollte ein halbes em reichen.

```
#lselect li a.c {display: inline;
  font: bold 1em Times, serif;
  letter-spacing: 0; text-transform: none;}
#larchives li, #lresources li {margin-bottom: 0.5em;}
</style>
```

Der andere Unterschied ist, dass die Links in diesem Abschnitt ein geringeres Schriftgewicht haben als die in der »Design List«. Idealerweise könnten wir sie auch extrem dünn machen, indem wir etwas wie `font-weight: 100;` wählen, aber das wird nicht sonderlich gut unterstützt. Stattdessen werden wir die Farben der Links etwas mehr mit dem Hintergrund verschmelzen lassen als die dunkelgrünen Links der »Design List«, indem wir ein helleres Grün wählen, um die visuelle Gewichtung zu mindern.

```
#larchives li, #lresources li {margin-bottom: 0.5em;}
#larchives li a, #lresources li a {color: rgb(126,164,139);}
</style>
```

Mit dieser kleinen farblichen Änderung – deren Ergebnis Sie in Abbildung 10.22 sehen können – sind wir am Ende unseres Stylesheets angekommen, das in Listing 10.4 dargestellt ist.

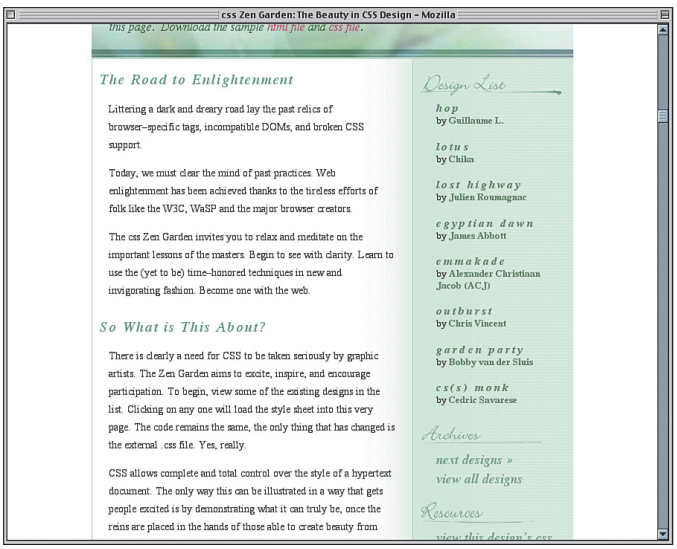


ABBILDUNG 10.22

Die Links für »Archive« und »Resources« werden mit geringerem visuellen Gewicht gestylt.

Listing 10.4 Das vollständige Stylesheet

```
body {margin: 0; padding: 0; text-align: center;
  color: #000; background: #FFF;}
acronym {border: none;}
a {text-decoration: none;}
a:link {color: rgb(179,63,96);}
a:visited {color: rgb(90,32,48);}
a:hover {text-decoration: underline;}
#container {width: 647px; margin: 75px auto 0; padding: 0;
  text-align: left; position: relative;}
#pageHeader {background: url(pageHeader.jpg) 0 0 no-repeat;
  height: 157px; width: auto;}
#pageHeader h1 {background: url(ph-flower.gif) 0 0 no-repeat;
  height: 330px; width: 250px; position: absolute; z-index: 101;
  top: -95px; right: -80px; margin: 0;}
#pageHeader h1 span {visibility: hidden;}
#pageHeader h2 {position: absolute; z-index: 102;
  top: 134px; right: 140px; margin: 0; padding: 0;
  color: rgb(91,131,104);
  text-transform: lowercase; letter-spacing: 0.2em;
  font: bold italic 1.1em/1em Times, serif; text-align: right;}
#pageHeader h2 acronym {text-transform: uppercase;}
#quickSummary {font: italic 1em/2 Times, "Times New Roman", serif;
  background: rgb(94%,98%,96%) url(quickSummary.jpg) 0 100% no-repeat;
  color: rgb(42,92,42);
  margin: 1px 0; padding: 1em 180px 1.5em 1.5em;}
```

```
#quickSummary p {display: inline;}
#quickSummary p.p2 {margin-left: 0.25em;}
#quickSummary p.p2:after {content: ".";}
#preamble, #supportingText {padding-right: 217px;}
#preamble p, #supportingText p {font-size: 90%; line-height: 1.66em;
   margin: 0 1.5em; padding: 0.5em 0;}
#preamble h3, #supportingText h3 {letter-spacing: 0.1em;
  font: italic 1.2em Times, "Times New Roman", serif;
  color: rgb(107,153,139); margin: 1em 0 0.5em 0.5em;}
#preamble {border-top: 1px solid rgb(184,214,194);
  background: url(side.jpg) 100% 100% repeat-y;}
#supportingText {background: url(side.jpg) 100% 0 repeat-y;}
#preamble, #explanation, #participation, #benefits, #requirements {
  border-left: 1px solid rgb(184,214,194);
  padding-top: 1px; padding-bottom: 1px;}
#requirements {border-bottom: 1px solid rgb(184,214,194);
  background: url(main.jpg) 100% 100% no-repeat;
  padding-bottom: 100px;}
#footer {margin: 0 -217px 0 0; height: 123px;
  background: #FFF url(footer.jpg) 100% 1px no-repeat;
  padding: 60px 1em 0 0.5em;}
#footer a {color: rgb(207,216,214); line-height: 1em;
  font-size: 1.25em; font-weight: 100;}
#linkList {position: absolute; z-index: 11;
  width: 216px; top: 157px; right: 0;
  margin-top: 8.6em;}
#linkList ul {margin: 0.5em 1em 0 2em; padding: 0;}
#linkList li {list-style: none;}
#linkList a {color: rgb(99,131,101);
  font: italic 1.15em Times, serif;
  text-transform: lowercase;}
#linkList h3 {margin: 1em 0 0; width: 216px; height: 35px;
  background: url(resources.gif) 10px 50% no-repeat;}
#lselect h3 {background-image: url(design-list.gif);}
#larchives h3 {background-image: url(archives.gif);}
#linkList h3 span {display: none;}
#lselect li {font-size: 85%; margin-bottom: 1.5em;}
#lselect li a {display: block; font-weight: bold;
  letter-spacing: 0.2em; text-transform: lowercase;}
#lselect li a.c {display: inline;
  font: bold 1em Times, serif;
  letter-spacing: 0; text-transform: none;}
#larchives li, #lresources li {margin-bottom: 0.5em;}
#larchives li a, #lresources li a {color: rgb(126,164,139);}
```

10.5 Ein PNG einfügen

Und PNG heißt ... ?

PNG steht für *Portable Network Graphics* und wird gewöhnlich als »Ping« ausgesprochen. Es handelt sich um ein in den frühen 90er Jahren erfundenes Bildformat, das 1996 eine W3C-Empfehlung erhielt (`http://www.w3.org/TR/REC-png-multi.html`). Es war als patentfreier und technisch überlegener Ersatz für das GIF-Format gedacht.

Sie erinnern sich ja noch, dass wir in unserem Projekt ganz früh dem Design das Bild einer großen Blüte hinzugefügt haben. Diese Bilddatei ist ein GIF89a mit einem transparenten Bereich (schauen Sie sich noch einmal die Abbildung 10.5 an, falls Sie sich nicht mehr genau erinnern). Wir hatten kurz erwähnt, dass ein PNG-Bild die deutlich schönere Lösung wäre und dass wir am Ende des Projekt noch mal darauf zurückkommen wollten.

Nun, jetzt befinden wir uns am Ende des Projekts und sprechen also über die Verwendung eines PNGs statt eines GIFs.

Aus Sicht des visuellen Designs ist der wichtigste Vorteil eines PNG, dass eine PNG-Datei Informationen zur Gammakorrektur beinhalten kann (damit Sie sich keine Sorgen darüber machen müssen, ob Ihre Bilder bei verschiedenen Betriebssystemen heller oder dunkler erscheinen) und auch einen Alpha-Kanal mit bis zu 16 Bit. Das erlaubt eine deutliche ausgefeiltere Transparenz und Transluzenz als die einfache An-Aus-Transparenz von GIF-Dateien. In einem GIF ist ein Pixel entweder opak oder transparent, da gibt es keine Übergänge. Bei einem PNG-Bild kann jedes Pixel in beliebigen Abstufungen semi-opak sein. Also ist es ein Kinderspiel, ein PNG zu erstellen, das einen fließenden Übergang von ganz schwarz oben bis völlig transparent am unteren Rand aufweist.

Wir wollen uns die Abbildung 10.23 anschauen, in der das von uns verwendete PNG mit den in der Datei enthaltenen Kanälen zu sehen ist.

ABBILDUNG 10.23

Das PNG-Bild und die darin enthaltenen RGBA-Kanäle

Die Rot-, Grün- und Blau-Kanäle (RGB) sind leicht nachvollziehbar. Der Alpha(A)-Kanal, derjenige ganz unten, zeigt die Transparenz-Maske für das Bild. Jeder Bereich des Kanals, der komplett schwarz ist, steht für vollständig transparent, und die völlig weißen Bereiche sind opak. Die ganzen Grauabstufungen repräsentieren semi-opake Bereiche. Das Bild selbst wird über einem weißen Hintergrund zusammengesetzt gezeigt.

Durch die Maske des Alpha-Kanals wird der Schatten durchscheinend, und der Bereich um die Blüte herum wird völlig transparent. Wir können allerdings das GIF nicht einfach durch dieses PNG ersetzen, weil der Hintergrund der Kopfzeile direkt an der Kante der Blüte aufhört (schauen Sie noch einmal in Abbildung 10.4 nach). Wir

brauchen einen neuen Hintergrund für den Masthead, der sich in diesem Design über den ganzen oberen Bereich erstreckt – so wie der in Abbildung 10.24.

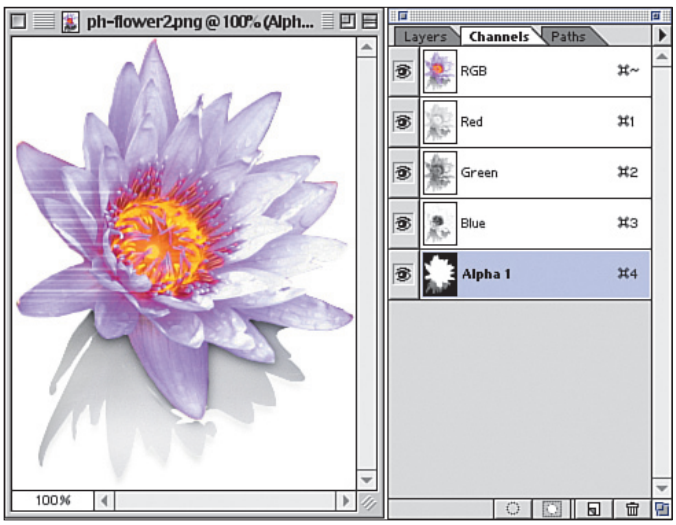


ABBILDUNG 10.24
Der neue, erweiterte Hintergrund des Mastheads

Wenn wir es ganz gründlich haben wollen, könnten wir den Rest des weißen Hintergrundbereichs mit Bildern füllen, aber weil die Blüte über diesem Bereich platziert sein wird, können wir das auch lassen. Wir wollen die alten Bilder durch den neuen Hintergrund (`pageHeader2.jpg`) und das PNG-Bild (`ph-flower2.png`) ersetzen.

```
#pageHeader {background: url(pageHeader2.jpg) 0 0 no-repeat;
  height: 157px; width: auto; }
#pageHeader h1 {background: url(ph-flower2.png) 0 0 no-repeat;
  height: 330px; width: 250px; position: absolute; z-index: 101;
  top: -95px; right: -80px; margin: 0;}
```

So wird das PNG in den meisten aktuellen Browsern seinen Zauber entfalten können. Eine krasse Ausnahme gibt es aber: den Internet Explorer für Windows! (Sogar der IE5/Mac behandelt das PNG korrekt.) Der IE/Win wird das PNG anzeigen, aber den Alpha-Kanal völlig außer Acht lassen. Das unterläuft natürlich unsere Absicht. Zum Glück können wir den IE/Win hacken, damit er den Alpha-Kanal über eine HTC-Datei (auch als Behavior-Datei bezeichnet) erkennt und darstellt. Indem wir für die Opazität des PNGs eine Behavior-Datei einsetzen (im Web gibt es eine Menge davon, obwohl die meisten sich auf den Umgang mit Inline-PNGs statt mit Hintergrund-PNGs konzentrieren) und sie mit unserem Stylesheet verknüpfen, erhalten wir das Ergebnis, das in Abbildung 10.25 gezeigt wird, und zwar sowohl bei Opera, Mozilla, Firefox, IE/Mac, Safari usw.

```
#pageHeader h1 {background: url(ph-flower2.png) 0 0 no-repeat;
  height: 330px; width: 250px; position: absolute; z-index: 101;
  top: -95px; right: -80px; margin: 0;
  behavior: url(png-opacity.htc);}
```

Abbildung 10.25

Über das eingefügte PNG erhalten wir einen zusätzlichen visuellen Glanz.

Wie wir in Projekt 6 besprochen haben, ist `behavior` kein Bestandteil von CSS, sondern ein CSS-ähnliches Statement, das proprietär dem Internet Explorer für Windows zu eigen ist. Wenn Sie also eine Deklaration mit `behavior` verwenden, wird das eine Validierung Ihres CSS verhindern. Es gibt keine Möglichkeit, das zu umgehen.

Es gibt einen weiteren potenziellen Nachteil bei der Verwendung eines PNG. Weil das Bild nicht nur die Farbinformationen, sondern auch den Alpha-Kanal enthält, wird es meist größer als andere Bildformate. Wenn das PNG 32-Bit-Farben verwendet, wozu es perfekt in der Lage ist, erhöht das die Dateigröße noch weiter; der Vorteil ist, dass das Bild anstatt bloß 256 oder 32.768 Farben einen Farbraum von mehreren Millionen Farben haben kann. PNG verwendet auch verlustfreie Komprimierung, und das kann zu größeren Dateien führen. Im Gegensatz dazu verwendet JPEG eine verlustbehaftete Kompression, und darum wird das Bild »knitterig«, wenn Sie ein JPEG sehr hoch komprimieren.

Wie sieht das aber mit echten Werten aus? Nun, die GIF-Version der Blüte ist 32 KB groß. Das mit Photoshop gespeicherte PNG ist 152 KB groß. Das ist eine deutlich höhere Dateigröße, bloß um einen durchscheinenden Schatten zu bekommen!

Es ist möglich, über verschiedene Tools die Dateigröße des PNG zu senken; Porter Glendinning, einer der Technischen Berater dieses Buches, konnte mit verschiedenen Tools das Blüten-PNG auf etwa 60 KB verkleinern. Das ist immerhin noch doppelt so groß wie das GIF. Ist es das wert? Vielleicht ja, vielleicht auch nicht. Aber dann hat Porter das Bild durch das Utility `pngquant` geschickt und es auf 28 KB gekriegt – etwa 4 KB *kleiner* als das Original-GIF, und es sieht außerdem noch viel, viel besser aus.

Und warum benutzt man PNGs dann nicht immer? Dafür gibt es zwei Gründe. Zum einen liegt es an der mangelhaften Unterstützung durch den IE/Win; die meisten Autoren kämen nie auf den Gedanken, dass eine per Stylesheet aufgerufene Behavior-Datei

Wo gibt es `pngquant`?

Sie können das Utility `pngquant` unter `http://www.libpng.org/pub/png/apps/pngquant.html` finden. Es ist für praktisch alle Betriebssysteme erhältlich – von Windows und Linux über Mac OS bis zu AmigaOS.

in der Lage wäre, den Support für ein Bildformat zu ermöglichen. (Also mir wäre das jedenfalls nicht eingefallen.)

Zum andern gehen die meisten kommerziellen Tools wie Photoshop mit PNGs sehr nachlässig und wenig intelligent um. Normalerweise wird Ihnen nur eine kompromisslose Wahl geboten, dass Sie also Ihr PNG entweder als eine 8-Bit-Farbdatei mit oder ohne Transparenz speichern können (im Grund ein GIF in einem anderen Format), und das heißt, die Datei bleibt klein, oder Sie entscheiden sich für die vollständige 24-Bit-Farbe mit einem 8-Bit-Alpha-Kanal, was zu richtig großen Dateien führt (da kommt einem der Wert 152 KB wieder in den Sinn). Wahrscheinlich haben PNGs deswegen den Ruf, sehr hübsch, aber auch unbeschreiblich aufgebläht zu sein. Der Fehler, so scheint es, liegt aber nicht im Format, sondern bei den Tools.

Wenn Autoren abgestufte Zwischenlösungen anwählen könnten (wie ein 8-Bit-Farbbild mit einem vollständigen Alpha-Kanal), dann kämen wir auch schon weiter. Momentan können Autoren über drei einfache Schritte in ihren Designs PNGs verwenden:

- Erstellen von PNGs mit vollständigen Alpha-Kanälen für die Verwendung im Web.
- Die resultierenden PNGs durch ein frei erhältliches Optimierungstool wie `pngquant` schicken.
- Eine HTC-Datei damit verknüpfen, um den IE/Win zum Mitspielen zu bewegen.

Das ist ein Schritt mehr als gewöhnlich, aber für ein besseres visuelles Grafikdesign und reduzierte Dateigrößen sind die Vorteile die kleine zusätzliche Mühe auf jeden Fall wert.

Der Mangel an breiter Unterstützung von PNGs im IE/Win und allgemein verwendeten Programmen hat deren Akzeptanz unzweifelhaft behindert, und so kennt man sich nicht so gut mit ihnen aus wie mit GIFs und JPEGs. Vielleicht reicht es zur Verbesserung dieser Situation, dass Autoren sich viel häufiger für PNGs entscheiden und sich gegenseitig darüber informieren, was sie dabei gelernt haben.

10.6 Betrachtungen

In vielerlei Hinsicht war dies eines der schwierigsten Projekte, die ich für dieses Buch geschaffen habe. Warum? Weil ich das Markup nicht verändern konnte und weil ich auf ein sehr spezielles visuelles Ziel hingearbeitet habe. Diese beiden Einschränkungen kombiniert haben den Arbeitsablauf sehr, sehr interessant gemacht.

Das perfekte Beispiel dafür ist die Positionierung der Seitenleiste. Klar ist es eine coole Sache, dass wir für die Platzierung der Link-Liste `top` und `margin-top` in

Indizierter Erfolg

In der letzten Phase der Arbeit an diesem Buch haben wir noch einen anderen Weg zu kleinen PNGs gefunden. Bevor das PNG in Photoshop gespeichert wird, konvertieren Sie das Bild zu *Indizierte Farben* und achten darauf, dass Sie die Option *Transparent* selektiert haben. Bei einem ersten Test mit dem Abspeichern einer Version von `ph-flower2.png` mit indizierten Farben erhielten wir eine Datei der Größe 43 KB, aber einem vollständig intakten Alpha-Kanal. Bei der Drucklegung dieses Buches waren noch keine Details darüber verfügbar, wie man PNGs am besten für das Webdesign nutzt, aber das wird sich hoffentlich ändern.

Kombination verwenden können, aber es hat sich als wackelige Lösung herausgestellt. Angenommen, der User setzt die Textgröße auf 120 % der Normalgröße. Der Text der Zusammenfassung verläuft dann sofort über vier Zeilen, und es gibt keine Möglichkeit für den `top`-Wert, sich als Folge davon anzupassen (außer bei einer Verwendung von JavaScript, aber das sprengt den Rahmen dieses Buches). Damit alles ausgerichtet bleibt, müsste er sich auf `10.6em` ändern. Stattdessen hat es dazu geführt, dass der Inhalt der Seitenleiste den Bereich der Zusammenfassung überlappt.

Dies kommt wie schon gesagt daher, dass das Markup des Dokuments verbotenes Terrain war. Wäre das nicht der Fall gewesen, hätte ich einige Sachen überarbeitet, damit der gesamte Inhalt der Hauptspalte in einem `div`-Element wäre, dem ich dann einfach nur einen einzigen Rahmen hätte geben können. Ich hätte auch die Link-Liste in dieses `div` gesetzt und es dann als Container-Block für die Link-Liste genommen. Dann hätte der User die Textgröße beliebig verändern können, und die Seitenleiste wäre immer noch passend zur Inhaltsspalte ausgerichtet geblieben. Und die Gefahr einer Überlappung wäre gebannt.

Dies beleuchtet ein Problem, das sehr wichtig ist, aber oft übersehen wird: *Die Darstellung hängt von der Struktur ab.* Sie kennen vielleicht den Ausdruck »Vollständige Trennung von Struktur und Darstellung«. Das ist mit der heutigen Technologie unmöglich, kann auch immer unmöglich bleiben, obwohl ich kein Prophet bin und nicht voraussagen kann, was in fünf, zehn oder dreißig Jahren möglich sein wird. Ich kann aber sagen, dass zum gegenwärtigen Zeitpunkt ein Dokument ohne jegliche Struktur – das heißt, keine Elemente und nur ein undifferenzierter Berg von Text – nicht auf sinnvolle Weise gestylt werden kann. Ohne Absätze, Überschriften, und Ankerelemente zum Markieren der Hyperlinks gibt es keine Hoffnung, dass die Sache schön aussehen wird.

Genauso gilt: Wenn die Struktur eines Dokuments nicht sonderlich gut mit dem gewünschten visuellen Ergebnis zusammenhängt (wie es in diesem Projekt der Fall war), dann müssen Sie schließlich entweder sehr kreativ werden oder doch etwas ändern. Gewöhnlich werden Designer einfach die Struktur ändern, damit sie besser zu ihren Layoutbedürfnissen passt. Das ist in Ordnung – tatsächlich ist das oft auch eine gute Idee. Die andere Möglichkeit ist, das visuelle Layout zu ändern – weg von dem, was Sie wollen, und hin zu dem, was die Struktur des Dokuments auch unterstützen kann. Das ist auch prima, obwohl es allgemein nicht so befriedigend ist.

Also sollten Sie immer daran denken, dass Ihr Design von der Dokumentstruktur abhängt. Das bedeutet gelegentlich mal, ein für Präsentationszwecke nötiges `div` oder `span` einzufügen. Solange Sie das nur bei Bedarf machen, brauchen Sie sich nicht den Kopf darüber zu zerbrechen. Wenn Sie sich regelmäßig dabei ertappen, dass Sie `span`s in (oder um) Links verschachteln, sollten Sie allerdings mal drüber nachdenken, was Sie da eigentlich gerade tun. Es ist wichtig, die Dinge so einfach und strukturell so angemessen wie möglich zu halten und dabei immer auf die eigenen Designanforderungen zu achten.

Eine letzte Bemerkung: Mein Respekt für die Designer, die Layouts für den Zen Garden geschaffen haben, ist durch dieses Projekt wesentlich gewachsen. Ein bekanntes Design an das Markup anzupassen, war eine echte Herausforderung; ein komplett neues und originelles Design auf diesem Markup aufzubauen, zeugt von unbeschreiblichem Geschick und Talent. Ich verneige mich in Ehrfurcht und Bescheidenheit vor jedem einzelnen der Designer des Zen Gardens. Dank geht an jeden Einzelnen von euch.

10.7 Spielwiese

Für dieses letzte Mal habe ich wirklich nur einen Vorschlag, um die Konzepte aus diesem Kapitel weiter zu entwickeln.

1. Schaffen Sie Ihr eigenes Design für den Zen Garden! Es muss nicht das wunderbarste Design aller Zeiten sein, und Sie brauchen es auch nicht einzureichen, obgleich Sie sicherlich ganz herzlich dazu eingeladen sind, wenn Sie es wollen. Sie brauchen nur ein oder mehrere Stylesheets zu kreieren, die die Basis-Datei des Zen Gardens auf eine neue und interessante Art präsentieren. Probieren Sie mit dem Markup verschiedene Layouts aus, gestalten Sie diese immer komplexer und sammeln Sie Ihre dabei Ihre Erfahrungen. Denken Sie daran: Das Markup soll nicht verändert werden. Sie können alles nur über CSS erreichen. Wenn Sie auf eine Begrenzung treffen, versuchen Sie, einen kreativen Weg darum herum zu finden. Probieren Sie Dinge aus, von denen Sie meinen, dass darin nicht die geringste Chance zum Funktionieren liegt. Wenn Sie einen Fehler machen, überlegen Sie einen Moment, bevor Sie alles rückgängig machen. War das Ergebnis vielleicht auf eine für Sie unerwartete Art interessant? Falls ja, dann folgen Sie doch diesem Pfad anstatt Ihrem ursprünglichen. Lassen Sie sich überraschen, welche interessanten Techniken oder Effekte Sie finden, wenn Sie einfach nur im Zen Garden arbeiten.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

W

Z