Markt+Technik

Birgit Kloss

CSS und DHTML

... für professionelle Einsteiger

digital studio one

Markt+Technik Verlag

CSS UND DHTML

FÜR WINDOWS UND MACINTOSH

Birgit Kloss

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei
Der Deutschen Bibliothek erhältlich.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt. Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1
06 05 04 03 02

ISBN 3-8272-6294-1

© 2002 by Markt+Technik Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, 81829 München/Germany
Alle Rechte vorbehalten
Einbandgestaltung: helfer grafik design, München
Lektorat: Jürgen Bergmoser, jbergmoser@pearson.de
Korrektorat: Christel Metke, Köln, c.metke@netcologne.de
Herstellung: Anna Plenk, aplenk@pearson.de
CD Mastering: Gregor Kopietz, gkopietz@pearson.de
Satz: mediaService, Siegen (www.media-Service.tv)
Druck und Verarbeitung: Kösel, Kempten, (www.KoeselBuch.de)

Printed in Germany

Inhaltsverzeichnis

Einführung **1**
Zu diesem Buch . . . 2
Welche Software braucht man? . . . 3
Die Browser . . . 4

Kapitel 1: CSS-Grundlagen **5**
CSS-Versionen . . . 6
CSS-Typen . . . 7
Der Aufbau eines Stils . . . 8
CSS-Methoden . . . 9
Angaben für die CSS-Werte . . . 10
Welche Stile für welche Tags? . . . 12

Kapitel 2: CSS in HTML einbinden **15**
Die lokale Methode . . . 16
Die interne Methode . . . 18
Die externe Methode . . . 20
Mehrere externe CSS-Dateien verwenden . . . 22
Externe CSS-Dateien importieren . . . 23
CSS-Dateien verschachteln . . . 24
Kombinieren der Methoden . . . 25
Klassen definieren . . . 26
Klassen verwenden . . . 27
IDs definieren . . . 28
IDs verwenden . . . 29
Unterklassen für HTML-Tags . . . 30
Gleiche Namen für verschiedene Unterklassen . . . 31
Formate für Pseudo-Elemente . . . 32
Formate für verschachtelte Tags . . . 33
Gruppen definieren . . . 34
Eigene Inline-Tags definieren . . . 35
Eigene Blocklevel-Tags definieren . . . 36
Tag-spezifische Eigenschaften . . . 38
Vererbte Tag-Eigenschaften . . . 39
Kommentare in CSS . . . 41
CSS für verschiedene Medien . . . 42
Seiten für den Druck optimieren . . . 43

Kapitel 3: Schriften 47
Schriften auf Webseiten 48
Die Schriftfamilie festlegen 52
Schriftgrößen festlegen 53
Kursive Schrift 55
Fette Schrift 57
Kapitälchen 58
Zeilenhöhen einrichten 59
Schriftwerte in Kurzschrift 61
Schriftfarben bestimmen 64

Kapitel 4: Text formatieren 65
Text horizontal ausrichten 66
Text vertikal ausrichten 68
Zeichenabstände festlegen 70
Wortabstände festlegen 71
Texteinzüge festlegen 72
Umbrüche und Leerzeichen 73
Textdekorationen 75
Groß-/Kleinschreibung 76

Kapitel 5: Listen, Pseudo-Formate und Maus-Cursor 77
Allgemeines zu Listen 78
Listen: Bullet-Typ festlegen 79
Listen: Bullet-Position 80
Listen: Grafiken für Bullets 82
Kurzschrift für Listen 83
Maus-Cursor nach Wahl 84
Pseudo-Elemente formatieren 86
Links formatieren 87
Abschnittsanfänge hervorheben 89
Klassen für Pseudo-Elemente 92

Kapitel 6: Elementhintergründe 95
Was sind Elemente? 96
Die Elementbox 97
Allgemeines zu Hintergründen 98
Netscape 4 und Hintergründe 99
Hintergründe in Inline-Tags 100
Hintergrundfarben 101
Hintergrundgrafiken mit CSS 102
Eine Hintergrundgrafik einfügen 103
Die Kachelung der Grafik 104
Scrollen der Grafik 105
Die Position der Grafik 106
Kurzschrift für den Hintergrund 108

Kapitel 7: Grössen, Rahmen, Abstände und Ränder 109
Elementgrößen . . . 110
Die Breite eines Elements . . . 111
Die Höhe eines Elements . . . 112
Abstände festlegen . . . 113
Rahmeneigenschaften . . . 116
Erlaubte Werte für Rahmen . . . 118
Rahmeneigenschaften festlegen . . . 119
Netscape 4 und Rahmen . . . 123
Ränder vs. Abstände . . . 124
Ränder festlegen . . . 125
Textumfluss um ein Element . . . 127
Textumfluss verhindern . . . 128

Kapitel 8: Elemente positionieren 129
Fenster-Grundlagen . . . 130
Arten der Positionierung . . . 131
Die Positionierungsart festlegen . . . 133
Die Position oben links festlegen . . . 134
Die Position unten rechts festlegen . . . 136
Die dritte Dimension . . . 138
Reihenfolge der Elemente . . . 139
Den z-index festlegen . . . 140
Eingebettete Elemente . . . 141
Absolute Elemente in relative Elementen einbetten . . . 142
Relative Elemente in absolute Elemente einbetten . . . 146

Kapitel 9: Anzeige von Elementen 149
display vs. visibility . . . 150
Anzeigeart eines Elements . . . 151
Sichtbarkeit von Elementen . . . 153
Sichtbarer Bereich eines Elements . . . 155
Was geschieht mit Überhängen? . . . 157

Kapitel 10: DHTML-Einführung 159
Was ist DHTML? . . . 160
Der Kampf der Giganten . . . 161
CrossBrowser DHTML . . . 162
Stärken und Schwächen . . . 163

Kapitel 11: Document Object Model 165
Was ist das DOM? . . . 166
Objekte erzeugen . . . 168
Was sind Event-Handler? . . . 169
Netscape 4 und Event-Handler . . . 170
Event-Handler einsetzen . . . 172
Mehrere Event-Handler verwenden . . . 173
Ein Event-Handler mit mehreren Werten . . . 174
So funktioniert das DOM . . . 175
Die Unterschiede der DOMs . . . 176
Die Problematik der drei DOMs . . . 180

Kapitel 12: Das CrossBrowser-DOM 181
Abfrage des DOM-Typs . . . 182
Die DOM-Funktion . . . 187
Einsatz des CrossBrowser-DOMs . . . 188
Netscape 4: eingebettete Ebenen . . . 190

Kapitel 13: Eigenschaften abfragen 193
Allgemeines zu den Funktionen . . . 194
Höhe und Breite von Objekten . . . 195
Die Positionen top und left . . . 197
Die Positionen rechts und unten . . . 199
Sichtbarkeitsstatus eines Objekts . . . 201
z-index eines Objekts . . . 204
Die Rechnerumgebung . . . 206
Den Browser abfragen . . . 207
Bildschirmabfragen . . . 210
Fensterabfragen . . . 212
Resize-Bug abfangen . . . 214

Kapitel 14: Einfache DHTML-Techniken 217
Allgemeines . . . 218
Objekte ein- und ausblenden . . . 219
Ein Objekt um einen bestimmten Wert verschieben . . . 222
Objekte in den Vordergrund holen . . . 225
Grafiken vorausladen . . . 228
Rollover-Effekte . . . 230
Ein neues Browserfenste röffnen . . . 232
Ein Fenster verschieben . . . 234
Ein Fenster schließen . . . 236

Kapitel 15: Weitere Techniken 237
Allgemeines zu Frames . . . 238
Das DOM für Objekte in Frames . . . 239
Tipps zu Frames . . . 243
Ein Objekt animieren . . . 246
Die Größe der Anzeigefläche ändern . . . 250
Drag&Drop von Objekten . . . 253

Kapitel 16: Navigation in der Website 263
Was macht eine gute Navigation aus? . . . 264
Vor-/Zurück-Links . . . 268
Navigation mit Auswahllisten . . . 269
Grafische Rollover-Leisten . . . 275
Dynamische Frames . . . 276
Dropdown-Menüs . . . 284
Fernsteuerung für die Website . . . 285
Die Website zum Buch . . . 291

Anhang A: Namenskonventionen 293
Namen vergeben . . . 294

Anhang B: CSS-Referenz 299
Die Referenztabellen . . . 300

Index 309

EINFÜHRUNG

Noch vor einigen Jahren diente das World Wide Web vor allem Akademikern und Journalisten dazu, Informationen auszutauschen. Das Aussehen einer Webseite spielte dabei eine untergeordnete Rolle, die Inhalte waren wichtiger.

Der Boom in der Computer- und Telekommunikationsindustrie läutete auch ein neues Web-Zeitalter ein. Günstige Preise für Computer, Webspace und Internetzugänge ermöglichten einem breiten Publikum, das Internet für sich „zu erobern". Die Anforderungen an Webseiten änderten sich; sie waren nicht mehr nur ein informelles Medium. Der Unterhaltungswert einer Webseite stieg und damit spielte auch das Design eine immer größere Rolle.

Diesen neuen Anforderungen wird HTML nur bedingt gerecht. HTML beschreibt die Struktur einer Webseite und stellt Befehle zur Verfügung, die das Setzen von Elementen wie Überschriften, Textabsätzen, Grafiken usw. erlauben. Doch die Kontrolle über das Aussehen einer Webseite ist in HTML sehr begrenzt. Die Webdesigner verlangten nach mehr Möglichkeiten bei der Gestaltung einer Webseite.

Diesen Wünschen trug das W3C (World Wide Web Consortium, eine Organisation, die technische Standards im WWW normiert) mit der Sprache CSS (Cascading Style Sheets) Rechnung.

Zu diesem Buch

Die Buchreihe *Digital Studio One* aus dem Hause Markt+Technik ist so aufgebaut, dass Sie sich schnell zurechtfinden.

Dieses Buch richtet sich an alle Web-Designer, die bereits Erfahrung mit HTML und JavaScript haben[1] und sich für ihre Webseiten mehr Layout-Möglichkeiten und Dynamik wünschen.

Das Buch ist für die Praxis geschrieben und erhebt keinen Anspruch auf Vollständigkeit. Vieles, was mit CSS theoretisch möglich ist, kann in der Praxis nicht umgesetzt werden, weil die Browser diese Möglichkeiten nicht unterstützen. Deshalb konzentriert sich das Buch hauptsächlich auf das, was Sie in der Praxis auch wirklich einsetzen können.

Sie lernen, welche Möglichkeiten CSS Ihnen bieten, auf das Aussehen einer Webseite Einfluss zu nehmen, und wie man sie effizient einsetzt. CSS bilden zusammen mit JavaScript die Grundlage für DHTML. Das Buch zeigt Ihnen, wie Sie DHTML browserübergreifend für Ihre Webseiten einsetzen können.

In den Anhängen finden Sie die Befehlsreferenz zu CSS, eine Liste mit Begriffen, die nicht für selbst vergebene Namen verwendet werden sollten, und natürlich das Stichwortverzeichnis zum Buch.

Die Website zum Buch

Zugunsten des Buchpreises wurde auf das Beifügen einer CD verzichtet. Dennoch können Sie sich das fehleranfällige Abtippen längerer Listings sparen.

Sie finden viele Beispiel-Listings auf meiner Supportseite zum Buch unter der URL *http://www.css-dhtml.de*.

Auf der Website steht auch eine (hoffentlich sehr kurze) Fehlerliste zur Verfügung. Weiterhin haben Sie die Möglichkeit, im Forum Fragen zu stellen.

Die CSS-Referenz, die Sie im Anhang dieses Buchs finden, steht ebenfalls online zur Verfügung und wird dort bei Erscheinen neuer Browserversionen aktualisiert.

1. Allen Leserinnen und Lesern, die mit HTML keine oder nur wenig Erfahrung haben, empfehle ich das Buch *HTML4*, ISBN 3-8272-5734-4.
   Einen leichten Einstieg in das Thema JavaScript finden Sie mit dem Buch *JavaScript*, ISBN 3-8272-5960-6.
   Beide Bücher sind in der Buchreihe *Digital Studio One* erschienen.

Welche Software braucht man?

Obwohl es mittlerweile nützliche Tools zum Schreiben von CSS gibt, benötigen Sie, wie auch bei HTML, lediglich einen einfachen Texteditor wie NotePad (Windows) oder SimpleText (Macintosh), die jeweils Bestandteil des Betriebssystems sind.

Von einer „ausgewachsenen" Textverarbeitung wie Word oder StarOffice ist abzuraten. Solche Programme fügen häufig unsichtbare Formatierungen ein, die zu Fehlern bei der Anzeige der Webseiten führen können.

Wenn Ihnen ein einfacher Texteditor zu unkomfortabel ist, sind textbasierte HTML-Editoren eine gute Alternative. Mit einem solchen Editor behalten Sie die volle Kontrolle über den Code und können trotzdem bequem arbeiten. Für fast jedes Betriebssystem gibt es solche Editoren mittlerweile zu günstigen Preisen oder sogar kostenlos als Freeware.

Was Sie außer einem Editor noch benötigen, ist ein CSS-fähiger Browser; mehr dazu erfahren Sie im nächsten Abschnitt.

✓ Tipp:

- Ohne andere Produkte herabsetzen zu wollen: Mein persönlicher Favorit für Windows ist der deutschsprachige HTML-Editor Phase5 von Ulli Meybohm. Phase5 ist Freeware und kann von der Internetseite des Autors kostenlos heruntergeladen werden: *http://www.meybohm.de/*

Die Browser

Die beiden bekanntesten Browser sind wohl Internet Explorer und Netscape. Der relativ neue Browser Opera erfreut sich in den letzten Monaten aber zunehmender Beliebtheit.

Wer sich mit HTML beschäftigt, weiß, wie unterschiedlich die Browser mit manchen Befehlen umgehen. Bei CSS und DHTML ist es leider nicht anders. Kein Browser unterstützt CSS vollständig, Opera 5 bietet laut W3C zurzeit die beste Unterstützung.

Einige Browser unterstützen CSS gar nicht, sie sind aber immer seltener anzutreffen. Diese Browser ignorieren die Stilregeln und zeigen die HTML-Befehle so an, als ob die Stile nicht vorhanden wären.

Wie eine Seite ohne die CSS aussieht, können und sollten Sie mit dem Browser Netscape 4.x überprüfen. Wenn man in den Einstellungen dieses Browsers JavaScript deaktiviert, funktionieren auch die CSS nicht mehr. Dies ist ein an sich unsinniges Verhalten, da CSS und JavaScript zwei verschiedene Standards sind. Beim Überprüfen der Seiten kommt uns diese Eigenheit aber sehr entgegen.

Wie die Seite des W3C mit und ohne CSS angezeigt wird, sehen Sie in Abbildung E.1 und Abbildung E.2.

Denken Sie beim Erstellen Ihrer Webseiten daran, dass viele Surfer immer noch mit älteren Browsern im Internet unterwegs sind, nicht mit den neuen 6er-Versionen. Setzen Sie daher als Standard Internet Explorer 5.x und Netscape 4.7x voraus und überprüfen Sie Ihre Webseiten stets mit beiden Browsern. Damit ist sichergestellt, dass möglichst viele Besucher Ihre Seiten wirklich so zu sehen bekommen, wie Sie sich das vorgestellt haben.

Abbildung E.1: Die Seite des W3C mit CSS

Abbildung E.2: Die gleiche Seite ohne CSS

✓ Tipp:

- Im Anhang finden Sie alle CSS-Regeln, auch diejenigen, auf die im Buch wegen der Inkompatibilität der gängigen Browser nicht ausführlich eingegangen wird. In dieser Liste können Sie nachsehen, welche Browser die verschiedenen CSS-Möglichkeiten unterstützen.

1

CSS-GRUNDLAGEN

HTML bietet Ihnen nur sehr begrenzt und mit viel Arbeitsaufwand die Möglichkeit, Elemente Ihrer Webseiten genau zu formatieren. Wenn Sie z.B. alle Überschriften in einer bestimmten Farbe darstellen wollen, müssen Sie diese Formatierung in jedem einzelnen Tag vornehmen. Das ist eine sehr mühselige und auch fehleranfällige Arbeit.

In einem modernen Textverarbeitungsprogramm haben Sie die Möglichkeit, zentrale Formate zu verwenden. Dort weisen Sie einem Absatz ein bestimmtes Format zu. Die Eigenschaften des Formats – Größe, Farbe und Schriftart – werden in einer zentralen Formatvorlage gespeichert. Wenn Sie die Eigenschaften in der Vorlage ändern, werden die Änderungen automatisch in den Dokumenten übernommen.

Diese Möglichkeit haben Sie mit CSS auch für Ihre Webseiten. CSS sind eine Ergänzung zu HTML, die es Ihnen ermöglicht, die Formateigenschaften von HTML-Befehlen zu definieren. Dabei können Sie die Formate für einzelne Tags, eine Seite oder zentral für die komplette Internetpräsenz definieren.

Das spart enorm viel Zeit und Arbeit, außerdem erledigt ein Computer die Formatierung ganz gewissenhaft. Sie müssen sich keine Gedanken mehr machen, ob Sie auch wirklich an allen Stellen das Format geändert haben.

Die Möglichkeiten beschränken sich aber nicht nur auf die Schriftformatierung. Sie können Webseiten mit Hintergrundbildern versehen, Elemente exakt positionieren und Einiges mehr.

CSS-Versionen

CSS werden wie HTML vom W3C ständig weiterentwickelt. Es existieren verschiedene Versionen, bei CSS werden sie Level genannt.

- *CSS Level 1 (CSS1)* wurde bereits 1996 vom W3C empfohlen. Diese Version enthält vor allem Möglichkeiten zur Formatierung von Schrift. Fast alle CSS1-Befehle werden von Internet Explorer 5.x und Netscape 4.x unterstützt.
- Während das W3C noch an der Nachfolgerversion von CSS1 arbeitete, wurde eine Möglichkeit zur Positionierung von Elementen benötigt. Aus diesem Grund stellte das W3C eine Übergangslösung zur Verfügung: *CSS-Positioning (CSS-P)*. Die Browserhersteller haben einen Teil von CSS-P in den 4er-Versionen integriert.
- *CSS Level 2 (CSS2)* wurde 1998 vom W3C empfohlen und ist derzeit offizieller Standard. CSS2 enthält alle Befehle der beiden Vorgängerversionen sowie einige weitere Möglichkeiten. Internet Explorer 5.x und Netscape 4.x unterstützen beide nur einen Teil von CSS2, während Opera 5 eine sehr weitreichende Unterstützung für CSS2 bietet.
- *CSS Level 3 (CSS3)* ist derzeit beim W3C in Bearbeitung, aber auch nach der Verabschiedung des neuen Standards wird es einige Jahre dauern, bis die Browser CSS3 unterstützen.

In diesem Buch finden zwar auch CSS2-Befehle Erwähnung, die von den gängigen Browsern nicht vollständig unterstützt werden, wir konzentrieren uns aber vor allem auf die Möglichkeiten, die in der Praxis wirklich eingesetzt werden können.

✓ Tipp:

- Webdesigner sind um Einiges offener gegenüber neuen Browserversionen als das breite Publikum. Viele Surfer im deutschsprachigen Raum sind in Bezug auf neue Browsersoftware eher konservativ eingestellt.
  Vergessen Sie nie, dass die Seiten bei *Ihren Besuchern* korrekt angezeigt werden sollen. Überprüfen Sie daher Ihre Webseiten immer mit den beiden gängigen Browsern Internet Explorer 5 (IE5) und Netscape 4 (N4). Die neuen 6er-Versionen sind dazu nicht geeignet, da sie bisher nur von relativ wenigen Surfern eingesetzt werden.

✓ Tipps:

- In HTML sind bei etlichen Tags sind die End-Tags optional. So wird z.B. das Tag `<p>` oft ohne das End-Tag `</p>` verwendet. In „reinem“ HTML ist das kein Problem. CSS funktionieren aber nur, wenn die End-Tags gesetzt werden.
- HTML-Tags sind nicht case-sensitiv, es ist im Grunde genommen also egal, ob Sie die Tags groß oder klein schreiben. In Hinblick auf den HTML-Nachfolger XHTML sollten Sie sich aber angewöhnen, alle Tags und Attribute klein zu schreiben. XHTML *ist* case-sensitiv, dort müssen die Befehle also zwingend klein geschrieben werden.
- Da Klassen und IDs bei Konflikten Vorrang vor den vorhandenen Formateigenschaften haben, können Sie bei der Verwendung von Selektoren bestimmte Tags anders formatieren. Beispiel:
  Sie haben den Selektor `p` in blauer, fetter Schrift formatiert und eine Klasse für rote Schrift angelegt. Wenn Sie einem bestimmten `<p>`-Tag die Klasse für rote Schrift hinzufügen, wird die blaue Farbe des `p`-Selektors außer Kraft gesetzt.

CSS-Typen

Es gibt verschiedene Möglichkeiten, CSS-Formate zu definieren:

- *HTML-Selektor*: Ein Selektor ist der Textteil eines HTML-Tags, also das was zwischen den spitzen Klammern steht. In CSS werden die Selektoren verwendet, um die Formateigenschaften des Tags zu bestimmen. Wenn Sie das Tag in Ihrer Seite verwenden, werden die definierten Formate automatisch auf die Tags angewendet.
  Beispiel:
  ```
  p { Stildefinition }
  ```
- *Klasse*: Mit Klassen können Sie Formate definieren, die Sie in mehreren Tags anwenden können. Die Namen der Klassen können Sie (unter Beachtung einiger Regeln) selbst festlegen. Die Formateigenschaften werden einem oder mehreren HTML-Tags lokal zugewiesen. Vor dem Namen der Klasse wird ein Punkt notiert.
  Beispiel:
  ```
  .KlasseName { Stildefinition }
  ```
- *ID*: Wie mit Klassen kann auch mit einer ID ein unabhängiges Format festgelegt werden. Eine ID wird aber normalerweise nur einmal im Dokument verwendet, z.B. um ein bestimmtes Element exakt zu positionieren. Vor dem Namen einer ID wird das Rautezeichen # notiert.
  Beispiel:
  ```
  #IDname { Stildefinition }
  ```

Klassen und IDs übernehmen das vorhandene Format eines Tags und fügen ihre Eigenschaften hinzu. Bei Konflikten haben sie jedoch Vorrang. Mehr zu den verschiedenen CSS-Typen erfahren Sie in Kapitel 2.

Der Aufbau eines Stils

Man spricht im Allgemeinen von einem Stil (engl. style) oder einer Stilregel. Ein Stil setzt sich aus einem Stilnamen[1] und einer oder mehreren Definitionen zusammen.

Der Stilname kann ein HTML-Selektor (`body`, `h1` usw.), eine Klasse oder eine ID sein. Die Definition legt fest, wie ein Element, auf das der Stil angewendet wird, angezeigt werden soll. Sie enthält eine Eigenschaft und einen Wert, die durch einen Doppelpunkt voneinander getrennt werden (siehe Abbildung 1.1).

Wenn mehrere Definitionen verwendet werden, müssen diese durch ein Semikolon voneinander getrennt werden. Bei nur einer Definition ist der Strichpunkt nicht zwingend erforderlich. Wenn aber später weitere Definitionen hinzugefügt werden, vergisst man leicht das Semikolon hinter der ersten Definition. Daher sollte man auch einzelne Definition immer mit dem Strichpunkt abschließen. (siehe Abbildung 1.2)

Bei internen und externen Stilen (siehe hierzu den Abschnitt *CSS-Methoden*) werden die Definitionen in die geschweiften Klammern gesetzt. Bei lokalen Stilen ist der Aufbau etwas anders (siehe Abbildung 1.3).

Abbildung 1.1: Ein Stil besteht aus einem Stilnamen und einer Definition. Die Definition setzt sich aus der Eigenschaft und einem Wert zusammen. Sie werden durch einen Doppelpunkt getrennt.

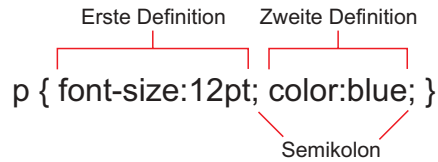

Abbildung 1.2: Wenn mehrere Definitionen verwendet werden, wird jede Definition mit einem Semikolon abgeschlossen.

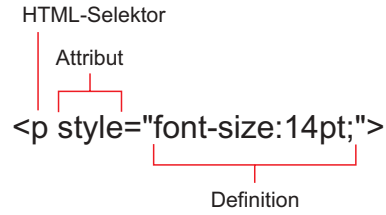

Abbildung 1.3: Ein lokaler Stil wird direkt im HTML-Tag gesetzt und durch das Attribut `style` eingeleitet. Die Definition ist der Wert des `style`-Attributs und steht daher in Anführungszeichen.

1. Die offizielle Bezeichnung des W3C für die Stilnamen lautet *selector*. Da aber auch von HTML-Selektoren die Rede ist, kann das leicht zu Verwechslungen führen. Deshalb wird in diesem Buch der Begriff *Stilname* oder schlicht *Name* verwendet.

```
<html>
<head>
<title>CSS-Methode</title>
<style type="text/css">
<!--
p {font-size:12pt;}
h1 {font-size:12pt;}
-->
</style>
</head>
```

Abbildung 1.4: Der Head-Bereich einer Webseite mit internen Stilregeln

Head-Bereich der HTML-Seite:

```
<html>
<head>
<title>CSS-Methode</title>
<link rel="stylesheet"
type="text/css"
href="stile.css">
</head>
```

Inhalt der Datei stile.css:

```
p {font-size:12pt;}
h1 {font-size:12pt;}
```

Abbildung 1.5: Der Head-Bereich einer Webseite mit externer CSS-Datei, darunter der Inhalt der Datei *stile.css*.

CSS-Methoden

CSS-Stile können mit drei Methoden in HTML eingebunden werden:

- *Lokal:* Direkt in einem HTML-Tag im Body der Webseite. Der Stil gilt dann nur für dieses Tag. Diese Methode wird auch *inline* genannt. Wie ein lokaler Stil definiert wird, sehen Sie in Abbildung 1.3.
- *Intern:* Im Head-Bereich einer Seite (siehe Abbildung 1.4). Die Stile gelten für die ganze Seite. Diese Methode wird auch als *eingebettet* bezeichnet.
- *Extern:* Die Stile werden in einer externen Datei mit der Endung *.css* gespeichert. Diese CSS-Datei wird in einer oder mehreren HTML-Seiten im Head-Bereich eingebunden (siehe Abbildung 1.5). Die Stile gelten für alle Webseiten, in denen die CSS-Datei eingebunden wird. Die externe Methode eignet sich hervorragend, um einer kompletten Website ein einheitliches Erscheinungsbild zu geben.

Sie können die Methoden auch kombinieren. Lokale Stile überschreiben dabei interne Stile und die internen Stile überschreiben Stilregeln, die in einer externen Datei definiert wurden. Diese Verhaltensweise hat den Cascading Style Sheets ihren Namen gegeben.

Wie die verschiedenen Methoden in den HTML-Dokumenten verwendet werden, erfahren Sie in Kapitel 2.

Angaben für die CSS-Werte

In den Stildefinitionen werden Werte angeben; die Einheiten dieser Werte sind abhängig von der verwendeten Eigenschaft.

Die Werte lassen sich in Gruppen aufteilen:

- *Beschreibende Angaben*
- *Farbangaben*
- *URL-Angaben*
- *Numerische Angaben*

Beschreibende Angaben

Es gibt etliche unterschiedliche beschreibende Angaben, wie z.B. `left`, `small` oder `lighter`. Welche beschreibende Angabe wo gemacht werden kann, ist immer abhängig von der Eigenschaft. Im Buch werden Sie bei allen Eigenschaften, die solche Werte verwenden können, entsprechende Hinweise finden.

Farbangaben

Farbangaben können auf verschiedene Arten gemacht werden (siehe Tabelle 1.1). Außer den festgelegten Farbnamen beschreiben alle Angaben, wie viele Rot-, Grün- und Blauanteile eine Farbe enthält. Standardmäßig werden hexadezimale Angaben verwendet.

URL-Angaben

URL-Angaben bezeichnen die Internetadresse einer beliebigen Datei. Man unterscheidet relative und absolute URLs (siehe Tabelle 1.2). Im Buch wird als Platzhalter für URLs häufig das Rautezeichen # verwendet, z.B. `<img src="#">`. In der Praxis ersetzen Sie dieses Zeichen dann durch die gewünschte URL.

Farbangaben r =rot, g =grün, b =blau	
`#rrggbb`	Hexadezimale Werte, z.B. `#ff33cc`. Das Rautezeichen leitet den Farbwert ein und ist zwingend erforderlich.
`rgb(r,g,b)`	In den Klammern werden numerische Werte (`0-255`) oder Prozentwerte (`0%-100%`) verwendet, z.B. `rgb(51,153,255)` oder `rgb(0%,20%,80%)`
`name`	Einer der 16 festgelegten Farbnamen, z.B. `blue` oder `red`

Tabelle 1.1: Mögliche Farbangaben

URL-Beispiele	
relativ	z.B. `src="seiten.css"`
absolut	z.B. `src="http://www.domain.de/bild.gif"`

Tabelle 1.2: Relative und absolute Adressen

✓ Tipps:

- Welche Werte für eine Eigenschaft erlaubt sind, können Sie im Anhang des Buchs in der CSS-Referenz nachlesen. Dort sind alle Eigenschaften mit den erlaubten Werten aufgelistet. Auch die unterschiedlichen beschreibenden Angaben wie `left`, `small` usw. sind in dieser Liste aufgeführt.
- Es gibt eine websichere Farbpalette, die 216 verschiedene Farben umfasst. Auf der Internetseite zum Buch finden Sie diese Farbpalette mit Angabe aller hexadezimalen Werte unter *http://www.css-dhtml.de.*

Absolute Maße	
pt	Punkt (=1/72 inch)
pc	Pica (=12 Punkte)
in	Inch (1in=2,54 cm)
mm	Millimeter
cm	Zentimeter

Tabelle 1.3: Absolute Maße

Relative Maße	
px	Pixel, abhängig von der Monitorauflösung
em	bezieht sich auf die Breite des Buchstabens m der verwendeten Schrift
ex	bezieht sich auf die Höhe des Buchstabens x der verwendeten Schrift

Tabelle 1.4: Relative Maße

Numerische Angaben

Die numerischen Angaben werden sehr häufig verwendet. Diese große Gruppe umfasst drei verschiedene Möglichkeiten:

1. *Zahlen* ohne Einheiten, die z.B. für Eigenschaften wie die Zeilenhöhe verwendet werden können.
2. *Prozentangaben*, die sich immer relativ zu bestimmten Elementen verhalten.
3. *Maße* sind Werte, die sich aus einer Zahl und einer bestimmten Maßeinheit zusammen setzen. Hier wird nach zwei Untergruppen unterschieden: absolute und relative Maße.
   - ▲ Absolute Maße haben feste Einheiten (siehe Tabelle 1.3).
   - ▲ Relative Werte sind abhängig von verschiedenen Faktoren, z.B. von der Monitorauflösung, der Browsersoftware oder der verwendeten Schrift. Die relativen Maße finden Sie in Tabelle 1.4.

Wenn keine Ganzzahl verwendet wird, muss als Trennzeichen der Punkt gesetzt werden. Also z.B. `1.2cm`, nicht `1,2cm`.

Bei Eigenschaften, für die numerische Angaben erlaubt sind, sollten Sie Pixel- oder Punktwerten den Vorzug geben. Damit werden die Elemente bei möglichst vielen Besuchern annähernd gleich groß angezeigt.

Verwendung numerischer Angaben

Maße sind fast überall als Werte erlaubt, Zahlen und Prozentangaben hingegen nicht. Wenn Sie in diesem Buch einen Hinweis finden, dass numerische Angaben erlaubt sind, bezieht sich das auf alle drei Möglichkeiten. Oft werden als erlaubte Werte aber nur die Maße angegeben. Zahlen und Prozentangaben können Sie dann nicht verwenden.

Welche Stile für welche Tags?

In welchen Tags welche Stileigenschaften verwendet werden können, erklärt sich häufig durch den Verwendungszweck. So macht es z.B. keinen Sinn, in einem Tag wie <b> eine Eigenschaft zu verwenden, die die erste Zeile eines Abschnitts einrückt.

Es gibt also grundsätzliche Unterschiede, die auf der Art der Tags beruhen. Wir unterscheiden *Blocklevel-Tags, Inline-Tags* und *Replaced-Tags*.

Blocklevel-Tags sorgen gewöhnlich dafür, dass eine neue Zeile begonnen wird. Sie können andere Blocklevel-Tags, Inline-Tags, Replaced-Tags und Text enthalten. Das bekannteste Blocklevel-Tag ist wahrscheinlich <p>, das einen neuen Absatz beginnt. Einige Blocklevel-Tags, die mit CSS formatiert werden können, sehen Sie in Tabelle 1.5.

Blocklevel-Tags	
body	Body der HTML-Seite
blockquote	Zitat definieren
center	Abschnitt zentrieren
div	Benutzerdefinierter Abschnitt
h1-6	Überschriften Größe 1 bis Größe 6
li	Listenpunkt
ol	geordnete Liste
p	Absatz
pre	vorformatierter Text
table	leitet eine Tabelle ein
td	Tabellenzelle
th	Tabellenkopf
tr	Tabellenreihe
ul	ungeordnete Liste

Tabelle 1.5: Blocklevel-Tags

Inline Tags	
a	Anker (für Links)
b	fett
big	stellt Text größer dar
em	betont (meist kursiv dargestellt)
font	Schrift festlegen
i	kursiv
small	stellt Text kleiner dar
span	benutzerdefiniertes Inline-Tag
strike	durchgestrichen
strong	stark betont (meist fett dargestellt)
sub	tiefgestellter Text
sup	hochgestellter Text
u	unterstrichen

Tabelle 1.6: Inline-Tags

Replaced-Tags	
img	Grafik einfügen
input	Formularfelder (verschiedene Typen)
object	Objekt einfügen
select	Auswahlliste im Formular
textarea	mehrzeiliges Formularfeld

Tabelle 1.7: Replaced-Tags

Inline-Tags beginnen keine neue Zeile, sie können auch keine Blocklevel-Tags enthalten, sondern nur andere Inline-Tags, Replaced-Tags und Text. Eine Auswahl an Inline-Tags finden Sie in Tabelle 1.6.

Replaced-Tags sind Tags, deren Höhe und Breite durch Attributwerte festgelegt werden kann. Sie können außerdem keinerlei andere Tags enthalten. Sie finden die Replaced-Tags in Tabelle 1.7. Netscape 4 unterstützt übrigens bei den Replaced-Tags keine lokalen Stile.

Die CSS-Referenz

Einige Stileigenschaften können nur für Blocklevel Tags angewendet werden, andere für alle drei Gruppen.

In der CSS-Referenz im Anhang des Buchs finden Sie die Liste der Eigenschaften. Dort ist auch immer mit angegeben, für welche Tag-Gruppe die Stile verwendet werden können.

Wenn Sie eine Eigenschaft auf ein Tag anwenden, die Formatierung aber im Browser nicht angezeigt wird, liegt dies oft daran, dass der verwendete Browser diese Eigenschaft nicht unterstützt. Welche Browser einen bestimmten Stil unterstützen, können Sie ebenfalls in der CSS-Referenz nachschlagen.

Tags und Container

Häufig ist von Tags die Rede, obwohl eigentlich ein HTML-Container gemeint ist.

Worin liegt der Unterschied und was ist eigentlich ein Container? Ein Tag ist ein HTML-Befehl der in den spitzen Klammern steht, z.B. `<img>` oder `<b>`. Zu vielen Tags gehört ein End-Tag, das gebildet wird, indem man den Slash / voranstellt: `</b>`. Start- und End-Tag bilden zusammen den Container: `<b>Containerinhalt</b>`.

Diese Container werden in der Praxis schlicht als Tags bezeichnet. Auch in diesem Buch ist das wird dieser Begriff verwendet. Wenn vom `<p>`-Tag die Rede ist, kann also der Container `<p>…</p>` gemeint sein.

2

CSS IN HTML EINBINDEN

Welche Methode Sie verwenden, um CSS in Ihre HTML-Seiten einzubinden, hängt davon ab, was Sie mit den Stilen erreichen möchten.

Geht es um eine einzige Seite, in der nur wenige Tags beeinflusst werden sollen oder möchten Sie bestehende Stile in einem bestimmten Tag überschreiben? Dann verwenden Sie die lokale Methode.

Eine einzelne Seite mit mehr Text und damit auch etlichen gleichlautenden Tags wie z.B. <p> formatiert man am besten mit der internen Methode. Sie beeinflusst nur die entsprechende Seite und spart Ihnen trotzdem viel Zeit.

Sobald mehr als eine Seite formatiert werden soll, ist der externen Methode der Vorzug zu geben. Wenn einzelne Seiten dann teilweise ein anderes Erscheinungsbild haben sollen, können Sie die externen Stile jederzeit durch die interne oder lokale Methode überschreiben.

Die lokale Methode

CSS bieten Ihnen die Möglichkeit, zentrale Formate zu definieren. Sie können aber weiterhin auch ganz gezielt auf das Erscheinungsbild eines einzelnen Tags Einfluss nehmen, indem Sie den Stil direkt im HTML-Tag einfügen. Damit überschreiben Sie z.B. Stilregeln, die mit der internen oder externen Methode definiert wurden.

Abbildung 2.1 zeigt die Syntax für einen lokalen Stil.

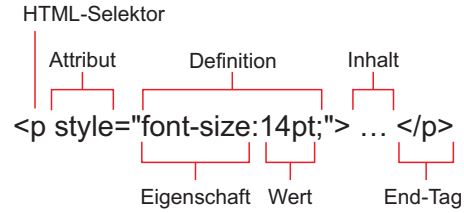

Abbildung 2.1: Die Syntax eines lokalen Stils

So fügen Sie einen Stil lokal in einem HTML-Tag ein:

1. `<p style=`
   Beginnen Sie das gewünschte HTML-Tag und fügen Sie nach dem HTML-Selektor ein Leerzeichen ein. Geben Sie dann `style=` ein.
2. `"font-size:12pt; color:#0000ff;"`
   In Anführungszeichen werden die Definitionen in der Form `Eigenschaft:Wert;` eingegeben. Schließen Sie jede Definition mit einem Semikolon ab.
3. `>Text, der im Absatz steht </p>`
   Schließen Sie das Tag und geben Sie dann den Text ein, der mit der Stilregel formatiert werden soll. Dann wird das End-Tag gesetzt.

```
<html>
<head>
<title>Lokale Methode</title>
</head>

<body style="background:#ffffff;
font-family:Arial,sans-serif;
font-size:10pt; color:#000000";>

<p style="font-size:16pt">Dieser Absatz
ist mit der Schriftgröße 16pt forma-
tiert.</p>

<p>In diesem Absatz ist kein lokaler Stil
vorhanden. Hier wird die Schriftgröße 10pt
verwendet, die im Body-Tag festgelegt
wurde.</p>

<p style="font-size:12pt">Dieser Absatz
ist mit der Schriftgröße 12pt forma-
tiert.</p>

</body></html>
```

Listing 2.1: Lokaler Einsatz der CSS...

Abbildung 2.2: ...und das Ergebnis im Browser

In Listing 2.1 sehen Sie den Quelltext einer Seite, in der CSS lokal eingesetzt werden. Im `<body>`-Tag sind die Hintergrundfarbe der Seite sowie die Schriftart, -größe und -farbe definiert. In den `<p>`-Tags sind teilweise andere Schriftgrößen festgelegt. Hier überschreibt ein lokaler Stil einen anderen lokalen Stil. Absätze ohne lokale Stilregeln übernehmen das Format des übergeordneten Tags, in diesem Fall also die Schriftdefinitionen des `<body>`-Tags.

Abbildung 2.2 zeigt das Ergebnis im Browser.

Zeilenumbrüche

Die Zeilenumbrüche im gezeigten Listing wurden wegen der Spaltenbreite im Buch gesetzt. In der HTML-Datei stehen die Definitionen in einer Zeile.

Die Zeilenumbrüche schaden aber nicht, wenn sie im Quelltext in der gezeigten Form tatsächlich gesetzt werden. Manchmal sind sie sogar sehr nützlich, da beim Einsatz von vielen Formaten der Quelltext so etwas übersichtlicher ist.

Die interne Methode

Bei der internen Methode werden die Stilregeln im Head-Bereich der Seite eingegeben. Sie gelten für das ganze Dokument, können bei Bedarf aber durch die lokale Methode überschrieben werden.

So fügen Sie Stilregeln mit der internen Methode ein:

1. `<style type="text/css"><!--`
   Setzen Sie den Cursor in den Head-Bereich und geben Sie `<style type="text/css"><!--` ein.
2. `p {`
   Beginnen Sie eine neue Zeile und geben Sie den gewünschten HTML-Selektor an. Dahinter folgt ein Leerzeichen und dann die öffnende geschweifte Klammer `{`.
3. `font-size:12pt; color:#0000ff; }`
   Geben Sie die gewünschten Definitionen in der Form `Eigenschaft:Wert;` ein. Jede Definition wird mit einem Semikolon abgeschlossen. Nach den Definitionen für dieses Tag muss die schließende geschweifte Klammer `}` gesetzt werden.
4. Wiederholen Sie die Schritte 2 und 3 für jedes Tag, das Sie formatieren möchten.
5. `--></style>`
   Beendet werden die Stilregeln durch die Eingabe von `--></style>`.

Abbildung 2.3 zeigt die Syntax eines internen Stils. Statt der HTML-Selektoren können auch Klassen oder IDs verwendet werden. Ebenso können Gruppen und Unterklassen von Tags definiert werden. Mehr dazu folgt später in diesem Kapitel.

Abbildung 2.3: Die Syntax eines Stils

✓ Tipps:

- Die meisten Browser erkennen CSS automatisch. Mit der Angabe von `type="text/css"` im `<style>`-Tag stellen Sie aber sicher, dass jeder CSS-fähige Browser auch wirklich erkennt, dass es sich um CSS handelt.
- Damit Browser, die nicht CSS-fähig sind, die Stilregeln nicht als anzuzeigenden Text auswerten, sollten Sie die Stilregeln immer in die HTML-Kommentarzeichen `<!--` und `-->` setzen. Dadurch ignorieren Browser, die keine CSS kennen, die Stilregeln komplett.

```
<html>
<head>
<title>Die interne Methode</title>

<style type="text/css">
<!--
h2 { font-family:Verdana,Arial,sans-serif;
font-size:16pt;}
p { font-family:Verdana,Arial,sans-serif;
font-size:12pt;}
-->
</style>

</head>
<body text="#000000" bgcolor="#FFFFFF">

<h2>Die interne Methode</h2>

<p>Bei der internen Methode werden die
Formatierungen im Head der Seite
notiert.</p>

<p style="font-size:10pt; font-style:ita-
lic;">Die Stile gelten für die ganze
Seite, bei Bedarf können einzelne Tags mit
der lokalen Methode formatiert werden.</p>
</body>
</html>
```

Listing 2.2: CSS intern eingesetzt...

In Listing 2.2 sehen Sie den Quelltext einer Seite, in der CSS intern eingesetzt werden. Im Head-Bereich sind für die Tags `h2` und `p` Stile definiert. Alle Tags im Quelltext übernehmen diese Formatierungen automatisch.

Durch Hinzufügen eines lokalen Stils können Sie für einzelne Tags die Stilregeln überschreiben oder neue Formate hinzufügen. Im letzten Absatz des Beispiels wurde auf diese Weise die Schriftgröße geändert und die Schrift kursiv gesetzt.

Abbildung 2.4 zeigt das Ergebnis des Beispiels im Browser.

Abbildung 2.4: ...und Wiedergabe im Browser

Die externe Methode

Bei der externen Methode werden die Stilregeln in einer separaten Datei mit der Endung *.css* gespeichert. Diese CSS-Datei wird dann in beliebigen HTML-Seiten im Head-Bereich eingebunden. Die Stile gelten für alle Seiten, in denen auf die CSS-Datei verwiesen wird. Sie können bei Bedarf durch die interne und die lokale Methode überschrieben werden.

So erstellen Sie Stilregeln in einer externen Datei:

1. `p {`
   In einer leeren Datei geben Sie den gewünschten HTML-Selektor an. Dahinter folgt ein Leerzeichen und dann die öffnende geschweifte Klammer `{`.
2. `font-size:12pt; color:#0000ff; }`
   Geben Sie die gewünschten Definitionen in der Form `Eigenschaft:Wert;` ein. Jede Definition wird mit einem Semikolon abgeschlossen. Nach den Definitionen für dieses Tag muss die schließende geschweifte Klammer `}` gesetzt werden.
3. Wiederholen Sie die Schritte 2 und 3 für jedes Tag, das Sie formatieren möchten.
4. `dateiname.css`
   Speichern Sie die Datei unter dem gewünschten Namen, z.B. `stile.css`

Listing 2.3 zeigt den Inhalt der Datei *stile.css*. Statt der HTML-Selektoren können auch Klassen oder IDs verwendet werden. Ebenso können Gruppen und Unterkategorien von Tags definiert werden. Mehr dazu folgt später in diesem Kapitel.

```
h2 {
font-family:Verdana,sans-serif;
font-size:18pt;
color:#0000c0;
}

h3 {
font-family:Verdana,sans-serif;
font-size:14pt;
color:#0000c0;
}

p {
font-family:Verdana,sans-serif;
font-size:10pt;
}
```

Listing 2.3: Der Inhalt der externen Datei *stile.css*. Die Definitionen für jedes Tag können in einer Zeile stehen. Übersichtlicher wird die Stildatei, wenn Sie die Stilnamen und die zugehörigen Definitionen in der gezeigten Form notieren.

CSS-Dateien

In einer CSS-Datei dürfen keine HTML-Tags stehen, auch nicht `<style>`. Die CSS-Datei enthält nur Angaben in CSS-Syntax.

```
<html><head>
<title>Die externe Methode</title>
<link rel="stylesheet" href="stile.css"
type="text/css">
</head>
<body text="#000000" bgcolor="#FFFFFF">
<h2>Die externe Methode</h2>
<h3>Die erste Seite</h3>
<p>In dieser Datei ist eine externe Stil-
datei eingebunden.</p>
</body></html>
```

Listing 2.4: Die erste Seite, in der die externe Stildatei eingebunden ist

```
<html><head>
<title>Die externe Methode</title>
<link rel="stylesheet" href="stile.css"
type="text/css">
</head>
<body text="#000000" bgcolor="#FFFFFF">
<h2>Die externe Methode</h2>
<h3>Die zweite Seite</h3>
<p>Auch in dieser Datei ist die externe
Stildatei eingebunden, die wir auch in der
ersten Seite verwendet haben.</p>
</body></html>
```

Listing 2.5: Die zweite Seite, die die externe Stildatei verwendet

So fügen Sie die externe Stildatei in ein HTML-Dokument ein:

1. `<link rel="stylesheet"`
   Setzen Sie den Cursor in den Head-Bereich der HTML-Seite und geben Sie `<link rel="stylesheet"` ein.
   Danach folgt ein Leerzeichen.
2. `href="dateiname.css"`
   Geben Sie `href="dateiname.css"` ein, gefolgt von einem Leerzeichen. Damit legen Sie fest, welche CSS-Datei Sie verwenden möchten. Im Beispiel heißt diese Datei *stile.css* und ist im gleichen Verzeichnis wie die HTML-Seite gespeichert. Wenn die Datei in einem anderen Verzeichnis gespeichert ist, achten Sie auf die richtige Pfadangabe. Sie könnte z.B. `href="../stile.css"` oder `href="verzeichnis/stile.css"` lauten.
3. `type="text/css">`
   Abgeschlossen wird der Befehl mit `type="text/css">` Ein End-Tag wird nicht gesetzt.

In Listing 2.4 und Listing 2.5 sehen Sie die Quelltexte von zwei Seiten, in denen die externe CSS-Datei eingebunden ist.

Mehrere externe CSS-Dateien verwenden

Sie können mehrere externe CSS-Dateien in einer Seite verwenden. Dazu notieren Sie im Head-Bereich der Seite einfach mehrere Link-Tags untereinander (siehe Listing 2.6). Dieses Vorgehen empfiehlt sich vor allem dann, wenn viele Formate definiert werden sollen. So halten Sie die CSS-Dateien übersichtlich, außerdem bedeuten kleinere Dateigrößen auch immer geringere Downloadzeiten.

Was passiert aber, wenn Sie in zwei CSS-Dateien jeweils den HTML-Selektor `h2` verwendet haben?

Die meisten Browser definieren die Stildateien in der Reihenfolge, in der sie im Head-Bereich eingebunden sind. Taucht ein HTML-Selektor in mehreren CSS-Dateien auf, wird normalerweise das Format verwendet, das in der zuletzt angesprochenen CSS-Datei definiert ist. In Listing 2.6 würde also das Format der Datei *stile2.css* verwendet werden, da sie nach *stile.css* angesprochen wird.

Achten Sie beim Einsatz mehrerer CSS-Dateien darauf, dass solche Konflikte erst gar nicht entstehen. Verwenden Sie daher einen Stilnamen nur in einer CSS-Datei, nicht in mehreren.

```
<html><head>
<title>Die externe Methode</title>
<link rel="stylesheet" href="stile.css"
type="text/css">
<link rel="stylesheet" href="stile2.css"
type="text/css">
</head>
<body text="#000000" bgcolor="#FFFFFF">
<h2>Die externe Methode</h2>
<h3>Die erste Seite</h3>
<p>In dieser Datei ist eine externe Stil-
datei eingebunden.</p>
</body></html>
```

Listing 2.6: Mehrere externe CSS-Dateien in einem HTML-Dokument verwenden

✓ Tipp:

- Ob und wie Sie die Stilregeln auf verschiedene CSS-Dateien verteilen bleibt Ihnen überlassen und sollte nach Verwendungszweck entschieden werden. Wenn Sie z.B. auf allen Seiten die Hintergrundfarbe und das Absatzformat einheitlich gestalten möchten, notieren Sie in einer CSS-Datei die entsprechenden HTML-Selektoren und binden diese CSS-Datei in allen Seiten ein. Auf einigen Seiten möchten Sie bestimmte Überschriftformate einsetzen; diese notieren Sie in einer zweiten CSS-Datei. Diese CSS-Datei binden Sie dann nur in die Seiten ein, die diese Formate wirklich verwenden sollen.

```
<html><head>
<title>
Import einer CSS-Datei
</title>
<style type="text/css">
<!--
@import url(stile.css);
@import url(stile2.css);
-->
</style>
</head>
<body>
...Seiteninhalte...
</body>
</html>
```

Listing 2.7: In dieser HTML-Seite wurden zwei CSS-Dateien importiert.

Externe CSS-Dateien importieren

Eine Möglichkeit zum Einbinden der externen CSS-Datei haben Sie bereits kennen gelernt:

```
<link rel="stylesheet" href="stile.css"
type="text/css">
```

Sie wird standardmäßig verwendet, da N4 nur diese Methode unterstützt. Dennoch möchte ich Ihnen die zweite Möglichkeit nicht vorenthalten: das Importieren von CSS-Dateien.

So importieren Sie eine CSS-Datei in eine HTML-Seite:

1. `<style type="text/css"><!--`
   Geben Sie im Head-Bereich Ihrer Seite `<style type="text/css"><!--` ein.
2. `@import url(dateiname.css);`
   In einer neuen Zeile schreiben Sie `@import url(dateiname.css);` Die URL-Angabe steht in runden Klammern. Im Beispiel ist die CSS-Datei im gleichen Verzeichnis gespeichert wie die HTML-Seite. Wenn die Datei in einem anderen Verzeichnis gespeichert ist, achten Sie auf die richtige Pfadangabe.
3. Wiederholen Sie Schritt 2 für jede CSS-Datei, die Sie importieren möchten.
4. `--></style>`
   Schreiben Sie in einer neuen Zeile `--></style>`; damit beenden Sie den Import der CSS-Datei.

Listing 2.7 zeigt die Syntax im Quelltext einer HTML-Seite. Um Benutzer des Browsers Netscape 4.x nicht auszuschließen, sollten Sie die `<link>`-Methode verwenden.

CSS-Dateien verschachteln

CSS-Dateien können Sie nicht nur in HTML-Seiten importieren, sondern auch in andere CSS-Dateien. Aber auch hier gilt: N4 versteht den Importbefehl nicht und zeigt die importierten Stile nicht an.

So importieren Sie eine CSS-Datei in eine andere CSS-Datei:

1. `@import url(stile.css);`
   In der CSS-Datei geben Sie `@import url(stile.css);` ein. Die URL-Angabe steht in runden Klammern. Im Beispiel sind die beiden CSS-Dateien im gleichen Verzeichnis gespeichert. Wenn die importierte Datei in einem anderen Verzeichnis gespeichert ist, achten Sie auf die richtige Pfadangabe.
2. Wiederholen Sie Schritt 1 für jede CSS-Datei, die Sie importieren möchten.

Listing 2.8 zeigt den Inhalt einer CSS-Datei, in der Stilregeln direkt definiert wurden. Zusätzlich wurden andere CSS-Dateien importiert. Die Importe können auch am Anfang der CSS-Datei stehen. Die Reihenfolge spielt lediglich in Konfliktsituationen eine Rolle (siehe hierzu den Abschnitt *Mehrere externe CSS-Dateien verwenden* weitere vorne in diesem Kapitel).

```
h2 {
font-family:Verdana,sans-serif;
font-size:18pt;
color:#0000c0;
}

h3 {
font-family:Verdana,sans-serif;
font-size:14pt;
color:#0000c0;
}

p {
font-family:Verdana,sans-serif;
font-size:10pt;
}

@import url(stile.css);
@import url(stile2.css);
```

Listing 2.8: Der Inhalt einer externen CSS-Datei, in die zusätzlich andere CSS-Dateien importiert wurden

```
<html><head>
<title>Kombination</title>
<link rel="stylesheet" href="stile.css"
type="text/css">
<style type="text/css">
<!--
h3 { font-style:italic;}
-->
</style>
</head>
<body text="#000000">
<h2 style="color:#00cc00;">Mit der lokalen
Methode formatiert</h2>
<h3>Die Eigenschaften der externen Datei
werden durch das Format der internen
Methode ergänzt.</h3>
<p>Für diesen Absatz gelten die Formate
der externen CSS-Datei, da weder ein loka-
ler noch ein interner Stil definiert
wurde.</p>
</body></html>
```

Listing 2.9: Hier sind die drei Methoden kombiniert.

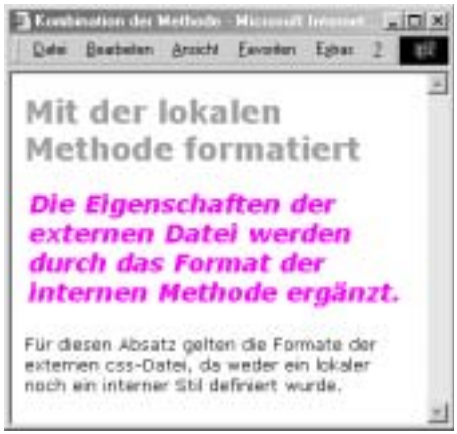

Abbildung 2.5: Das Ergebnis von Listing 2.7

Kombinieren der Methoden

Dass mit der lokalen Methode die internen und externen Stile überschrieben oder ergänzt werden können, haben Sie bereits in den vorangegangenen Abschnitten gesehen.

Ebenso ist es möglich, die interne und externe Methode zu verwenden. Dabei überschreiben die internen Stile die Regeln der externen Datei.

Diese Kombination ist immer dann sinnvoll, wenn Sie allen Seiten Ihrer Internetpräsenz ein einheitliches Erscheinungsbild geben möchten, in einer einzelnen Seite aber teilweise andere Formate verwenden möchten.

In Listing 2.9 sehen Sie die Kombination der drei Methoden. Die Überschriften `h3` übernehmen die Formate der externen Datei *stile.css* (siehe Listing 2.3) und werden durch den internen Stil zusätzlich kursiv formatiert. Die Überschrift `h2` verwendet die lokale Methode, d.h., mit der definierten Stilregel für die Textfarbe wird die entsprechende Eigenschaft der externen Datei überschrieben. Für die Absätze ist weder ein interner noch ein lokaler Stil angegeben. Sie übernehmen die Formate der externen Datei.

Abbildung 2.5 zeigt das Ergebnis im Browser.

Klassen definieren

Erinnern Sie sich noch an den Abschnitt über CSS-Typen in Kapitel 1? Dort wurde bereits erwähnt, dass für Stile nicht nur HTML-Selektoren verwendet werden können, sondern auch Klassen und IDs, mit denen Sie unabhängige Formate definieren können. Mit den IDs beschäftigen wir uns später noch; schauen wir uns zuerst die Klassen an.

Klassen können Sie sowohl in internen als auch externen Stilregeln verwenden, die Syntax bleibt dabei gleich (siehe Abbildung 2.6).

So definieren Sie eine Klasse:

1. `.rot`
   Eine Klasse wird immer von einem Punkt eingeleitet, denn sonst funktioniert die Stilregel nicht. Direkt dahinter folgt der Name der Klasse (siehe Tipp), gefolgt von einem Leerzeichen.
2. `{ font-size:12pt; }`
   In geschweiften Klammern werden die Definitionen in der Form `Eigenschaft:Wert;` notiert. Denken Sie daran, jede Definition mit einem Semikolon abzuschließen.
3. Wiederholen Sie die Schritte 1 und 2 für jede Klasse, die Sie definieren möchten.

Listing 2.10 zeigt einige Klassen, die in einer externen CSS-Datei definiert wurden. Bei der internen Methode werden die Regeln wie gewohnt von `<style> </style>` umschlossen.

Abbildung 2.6: Die Syntax einer Klasse. Der Stilname entspricht hier dem Namen der Klasse. Der Punkt gehört bei Klassen zum Stilnamen; ohne ihn funktioniert die Stilregel nicht.

```
.rot { color:#ff0000;}
.grün { color:#00ff00;}
.blau { color:#0000ff;}
p { font-size:12pt; }
```

Listing 2.10: Hier sind verschiedene Klassen für Schriftfarben definiert, darunter steht eine Stilregel für das HTML-Tag `<p>`.

Klassennamen

Den Namen der Klasse legen Sie selbst fest. Verwenden Sie nur Kleinbuchstaben, keine Umlaute, Leer- oder Sonderzeichen. Ziffern können Sie ebenfalls verwenden, der Name muss aber mit einem Buchstaben beginnen. Benutzen Sie keine Namen, die für JavaScript reserviert sind (siehe Anhang).

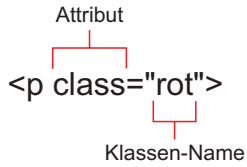

Abbildung 2.7: Die Klasse "rot" wird auf das <p>-Tag angewendet.

```
<html><head>
<title>Klassen</title>
<style type="text/css">
<!--
.rot { color:#ff0000;}
.grün { color:#00ff00;}
.blau { color:#0000ff;}
p { font-size:12pt; }
-->
</style>
</head><body bgcolor="#FFFFFF">
<h2 class="grün">Eine grüne Überschrift h2
</h2>
<h3 class="rot">Eine rote Überschrift h3
</h3>
<p class="rot">Ein Absatz in roter
Schrift.</p>
<p class="blau">Und nun ein Absatz in
blauer Schrift.</p>
</body></html>
```

Listing 2.11: Verschiedene Schriftfarben, die mit der internen Methode definiert werden.

Klassen verwenden

Die definierten Klassen werden einem HTML-Tag mit dem Attribut `class` hinzugefügt. Die Syntax sehen Sie in Abbildung 2.7.

So wenden Sie eine Klasse an:

1. `<p class=`
   Beginnen Sie das gewünschte HTML-Tag und fügen Sie nach dem HTML-Selektor ein Leerzeichen ein. Geben Sie dann `class=` ein.
2. `"klasse"`
   Schreiben Sie in Anführungszeichen den Namen der Klasse. Beachten Sie, dass hier der Punkt nicht verwendet wird.
3. `>Text, der im Absatz steht </p>`
   Schließen Sie das Tag und geben Sie dann den Text ein, der mit der Stilregel formatiert werden soll. Dann wird das End-Tag gesetzt.

Es ist wichtig, dass der Punkt, der die Klasse in der Stilregel einleitet, beim Formatieren des Tags nicht verwendet wird.

In Listing 2.11 sehen Sie den Quelltext einer Seite, in der verschiedene Schriftfarben mit der internen Methode definiert wurden. Die Farben wurden auf verschiedene Tags angewendet.

IDs definieren

Auch mit IDs können Sie unabhängige Formate definieren, die mit der internen oder externen Methode in eine HTML-Seite eingebunden werden.

Im Gegensatz zu den Klassen wird eine ID gewöhnlich aber nur einmal verwendet, um ein bestimmtes Objekt zu identifizieren. In DHTML sind IDs der Dreh- und Angelpunkt, da mit ihrer Hilfe JavaScript auf bestimmte Objekte angewendet werden kann. Die Syntax für eine ID-Regel sehen Sie in Abbildung 2.8.

Abbildung 2.8: Syntax einer ID-Regel

So definieren Sie eine ID:

1. `#name`
   Eingeleitet wird eine Stilregel für eine ID immer mit dem Rautezeichen #, dahinter folgt der Name, den Sie selbst festlegen.
2. `{ color:#ff0000; }`
   In geschweiften Klammern werden die Definitionen in der Form `Eigenschaft:Wert;` notiert. Denken Sie daran, jede Definition mit einem Semikolon abzuschließen.

Listing 2.12 zeigt einige IDs, die in einer externen CSS-Datei definiert wurden. Bei der internen Methode werden die Regeln wie gewohnt von `<style> </style>` umschlossen.

```
#bereich1 { color:#ff0000; }
#bereich2 { color:#00ff00; }
#bereich3 { color:#0000ff; }
```

Listing 2.12: Syntax für eine ID-Regel

ID-Namen

Den Namen der ID legen Sie selbst fest. Verwenden Sie nur Kleinbuchstaben, keine Umlaute, Leer- oder Sonderzeichen. Ziffern können Sie ebenfalls verwenden, der Name muss aber mit einem Buchstaben beginnen. Benutzen Sie keine Namen, die für JavaScript reserviert sind (siehe Anhang).

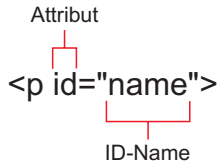

Abbildung 2.9: Die ID wird auf das <p>-Tag angewendet.

```
<html><head>
<title>Klassen</title>
<style type="text/css">
<!--
.rot { color:#ff0000;}
#bereich1 { color:#0000ff; }
#fett { font-weight:bold; }
p { font-weight:normal; }
-->
</style>
</head><body bgcolor="#FFFFFF">
<p class="rot" id="bereich1">
Ein Absatz in roter Schrift? <br>
Nein, in blauer Schrift, da die ID Vorrang
hat.</p>
<p style="color:#ff0000;" id="fett">
Und nun ein Absatz in fetter, roter
Schrift.</p>
</body></html>
```

Listing 2.13: Eine HTML-Seite, in der Klassen und IDs kombiniert werden

Abbildung 2.10: Darstellung im Browser

IDs verwenden

Die definierten IDs werden einem HTML-Tag mit dem Attribut `id` hinzugefügt. Die Syntax sehen Sie in Abbildung 2.9.

So wenden Sie eine ID an:

1. `<p id=`
   Beginnen Sie das gewünschte HTML-Tag und fügen Sie nach dem HTML-Selektor ein Leerzeichen ein. Geben Sie dann `id=` ein.
2. `"IDname"`
   Schreiben Sie den Namen der ID in Anführungszeichen. Beachten Sie, dass hier das Rautezeichen nicht verwendet wird.
3. `>Text, der im Absatz steht </p>`
   Schließen Sie das Tag und geben Sie dann den Text ein, der mit der Stilregel formatiert werden soll. Dann wird das End-Tag gesetzt.

Es ist wichtig, dass das Rautezeichen, das die ID in der Stilregel einleitet, beim Formatieren des Tags nicht verwendet wird.

IDs können in den Tags mit Klassen und lokalen Stilen kombiniert werden (siehe Listing 2.13). Bei Konflikten haben IDs immer den Vorrang vor allen anderen Stilregeln. Abbildung 2.10 zeigt das Ergebnis des Beispiels im Browser.

Unterklassen für HTML-Tags

Für HTML-Tags können Sie Unterklassen festlegen, die den vorhandenen Formaten weitere Eigenschaften hinzufügen. So können Sie z.B. mit dem HTML-Selektor festlegen, dass alle <p>-Tags eine bestimmte Schriftfamilie und Größe verwenden, in Unterklassen definieren Sie verschiedene Schriftgrößen für einzelne Absätze.

Das könnten Sie zwar auch mit der lokalen Methode realisieren, Sie müssten dann aber jedes Tag einzeln ändern, wenn Sie später andere Eigenschaften verwenden wollen.

Unterklassen ermöglichen es also, Tags unterschiedlich zu formatieren und trotzdem die Vorzüge der zentralen Formate zu nutzen.

In den Tags wird nur der Name angegeben, der in der Definition hinter dem Punkt steht. Wenn die Unterklasse also `p.klein` lautet, wird als Wert des `class`-Attributs `klein` angegeben (siehe Listing 2.14).

```
<html><head>
<title>Klassen</title>
<style type="text/css">
<!--
p { font-size:10pt;
font-family:Verdana, sans-serif;}
p.klein { font-size:8pt;}
p.gross { font-size:14pt;}

-->
</style>
</head><body bgcolor="#FFFFFF">
<p>Dieser Absatz verwendet die Eigenschaften, die für den HTML-Selektor p angegeben wurden.</p>
<p class="klein">Dieser Absatz verwendet die Unterklasse klein.</p>
<p class="gross"> Dieser Absatz verwendet die Unterklasse gross.</p>
</body></html>
```

Listing 2.14: Hinzufügen weiterer Eigenschaften mit Unterklassen

```
<html><head>
<title>Klassen</title>
<style type="text/css">
<!--
h2.gross { font-size:24pt;}
p.gross { font-size:15pt;}
-->
</style>
</head><body bgcolor="#FFFFFF">
<h2 class="gross">Die Unterklasse gross
für das h2-Tag.</p>
<p class="gross"> Dieser Absatz verwendet
die Unterklasse gross für das p-Tag. Wel-
che Stilregel verwendet wird, legt der
Browser anhand des Tags fest.</p>
</body></html>
```

Listing 2.15: Zwei Unterklassen mit gleichem Namen

Abbildung 2.11: Unterklassen mit gleichem Namen

Gleiche Namen für verschiedene Unterklassen

Sie können die Namen für Unterklassen mehrfach verwenden. Es ist z.B. möglich, die Unterklassen `p.gross` und `h2.gross` zu definieren. In beiden Tags wird dann `class="gross"` als Attribut angegeben. Die Browser unterscheiden anhand des Tags, welche Stilregel verwendet werden soll.

In Listing 2.15 sehen Sie zwei Unterklassen mit gleichem Namen; Abbildung 2.11 zeigt das Ergebnis im Browser.

Formate für Pseudo-Elemente

Pseudo-Elemente sind Teile einer Webseite, die sich nicht durch ein eindeutiges HTML-Tag ausdrücken lassen, z.B. "der erste Buchstabe", "die erste Zeile" oder "ein besuchter Link".

Welche Pseudo-Elemente es gibt, sehen Sie in Tabelle 2.1. Sie lernen diese Elemente in Kapitel 5 noch näher kennen. Hier geht es zunächst um die spezielle Syntax für solche Pseudo-Elemente.

Es gibt weitere Pseudo-Formate, z.B. einige Befehle für Drucklayouts, die aber von den gängigen Browsern noch nicht unterstützt werden und hier daher keine Erwähnung finden.

Element	Bedeutung
`a:link`	Hyperlink zu noch nicht besuchten Seiten
`a:visited`	Hyperlink zu bereits besuchten Seiten
`a:hover`	Hyperlink, der mit der Maus überfahren wird
`a:active`	Hyperlink, der gerade angeklickt wird
`:first-letter`	Erstes Absatzzeichen
`:first-line`	Erste Absatzzeile

Tabelle 2.1: Manche Pseudo-Elemente sind einem bestimmten Tag zugeordnet, z.B. `a:link`. Andere Elemente können in diversen Tags verwendet werden, sie stehen in der Tabelle ohne HTML-Selektor vor dem Schlüsselwort.

So definieren Sie ein Pseudo-Format:

1. `h2`
   Geben Sie den HTML-Selektor an, auf den das Pseudo-Format angewendet werden soll. `h2` ist hier nur ein Beispiel.
2. `:first-letter`
   Dahinter geben Sie einen Doppelpunkt an, gefolgt von dem Elementnamen. Nach dem Elementnamen folgt ein Leerzeichen.
3. `{ font-size:20pt; }`
   Notieren Sie nun wie gewohnt die gewünschten Stildefinitionen.

Zwei Beispiele für Pseudo-Formate:

```
a:link { color:#ff0000; }
```

```
p:first-letter { color:#0000ff; }
```

```
<html><head>
<title>Abhängige Formate</title>
<style type="text/css">
<!--
h3 { font-family:Verdana;}
h3 i { color:#ff0000;}
p { font-family:Verdana;}
-->
</style>
</head>
<body bgcolor="#FFFFFF">
<h3>Überschrift h3 ohne Kursivschrift</h3>
<h3>Überschrift h3 mit <i>kursivem Text,</
i> der rot dargestellt wird.</h3>
<p>In diesem Absatz steht <i>kursive
Schrift,</i> die nicht farbig ist.</p>
</body></html>
```

Listing 2.16: Verschachteltes Format und...

Abbildung 2.12: ...Darstellung im Browser

Formate für verschachtelte Tags

Neben den Unterklassen für HTML-Tags können Sie auch spezielle Formate für verschachtelte Tags definieren. Angenommen, Sie möchten in `h2`-Überschriften kursive Schrift verwenden, die eine andere Farbe hat als alle anderen kursiven Textpassagen. Hierzu verwenden Sie ein verschachteltes Format.

So definieren Sie ein verschachteltes Format:

1. `h3 i`
   Als Stilname geben Sie den HTML-Selektor des äußeren Tags an, gefolgt von einem Leerzeichen und dem HTML-Selektor, der das definierte Format erhalten soll. In unserem Beispiel ist das `h3 i`. Damit werden alle `<i>`-Tags formatiert, die innerhalb von `<h3>`-Tags stehen. Andere Textpassagen, die mit `<i>` kursiv formatiert werden, sind von der Formatierung also nicht betroffen.
2. `{ color:#ff0000; }`
   In geschweiften Klammern geben Sie die Definition für das innere Tag an.

In Listing 2.16 sehen Sie ein Beispiel; Abbildung 2.12 zeigt das Ergebnis im Browser.

Gruppen definieren

Wenn verschiedene Tags gleiche Eigenschaften besitzen sollen, können Sie diese Tags in Gruppen zusammenfassen.

So definieren Sie eine Gruppe:

1. `p,`
   Schreiben Sie den ersten Stilnamen auf, gefolgt von einem Komma.
2. `.blau,`
   Dahinter notieren Sie den nächsten Namen. Wenn weitere Selektoren folgen sollen, schreiben Sie wieder ein Komma.
3. Wiederholen Sie Schritt 2 für alle gewünschten Stilnamen. Nach dem letzten Selektor steht kein Komma mehr.
4. `{ color:#0000ff; }`
   In geschweiften Klammern folgen die Definitionen.

In Listing 2.17 sehen Sie ein Beispiel. Klassen und IDs können ebenfalls in einer solchen Gruppe stehen.

```
p,td,.blau { color:#0000ff;
font-family:Arial,sans-serif; }
```

Listing 2.17: Alle Stilnamen in der Gruppe erhalten die definierten Eigenschaften.

✓ Tipps:

- Gruppen können viel Zeit sparen, da die Eigenschaften für mehrere Tags zentral festgelegt und später geändert werden können. Bedenken Sie aber, dass Sie einem HTML-Selektor, der in einer Gruppe steht, nur über Unterklassen weitere Eigenschaften, die nicht die ganze Gruppe betreffen, zuweisen können.
- Legen Sie die Gruppierungen und die entsprechenden Eigenschaften nach dem Verwendungszweck fest. Es macht z.B. wenig Sinn, in einer Gruppe die Textgröße festzulegen, wenn HTML-Selektoren für Überschriften und Fließtext zu der Gruppe gehören, da diese normalerweise unterschiedliche Schriftgrößen haben.

```
<html><head>
<title>Eigene Inline-Tags</title>
<style type="text/css">
<!--
.rot { color:#ff0000;}
span.blau { color:#0000ff;}
p { font-weight:normal; }
-->
</style>
</head><body bgcolor="#FFFFFF">
<p>Im Absatz <span class="rot">steht rote
Schrift</span>, die mit einem benutzerde-
finierten Inline-Tag formatiert wurde.</p>
<p class="blau">Blaue Schrift? Nein, weil
die Klasse span.blau nur für das span-Tag
gilt.</p>
<p><span style="color:#00a000">Aber nun in
grün</span>, weil innerhalb des p-Tags das
span-Tag mit einem lokalen Stil steht.
</p>
</body></html>
```

Listing 2.18: Einige Inline-Tags

Abbildung 2.13: Darstellung im Browser

Eigene Inline-Tags definieren

Wenn man mitten in einem Fließtext Textpassagen mit CSS formatieren möchte, stellt sich die Frage, welches Tag man dazu verwenden soll.

Mit dem Tag `<span>` haben Sie die Möglichkeit, solche Formate zu realisieren. Die Stile werden wie gewohnt als Klassen oder IDs mit der internen oder externen Methode definiert. Sie können aber auch die lokale Methode einsetzen. Wenn Sie das `<span>`-Tag ohne Stil einsetzen, hat es keinerlei Auswirkungen, da es keine eigenen Eigenschaften besitzt. Das End-Tag `</span>` muss gesetzt werden.

So definieren Sie eigene Inline-Tags:

1. `<span class="name">`
   `<span id="name">`
   `<span style="Definition">`
   Je nachdem, ob Sie eine Klasse, eine ID oder einen lokalen Stil anwenden möchten, setzen Sie im `<span>`-Tag das entsprechende Attribut und den dazugehörigen Wert.
2. `Text`
   Schreiben Sie den Text, der formatiert werden soll.
3. `</span>`
   Dann wird das End-Tag gesetzt. Es ist zwingend erforderlich.

Listing 2.18 zeigt den Quelltext einer Seite, in der mehrere `<span>`-Tags gesetzt wurden.
In Abbildung 2.13 sehen Sie das Ergebnis im Browser.

Eigene Blocklevel-Tags definieren

Mit CSS haben Sie auch die Möglichkeit, ganze Textabschnitte, so genannte Blöcke, frei zu definieren.

Dazu wird das Tag verwendet. Auch hier können Klassen, IDs oder lokale Stile verwendet werden. Wie bei einem Blocklevel-Tag üblich, setzt auch das -Tag eine neue Zeile vor und hinter den Abschnitt, ohne allerdings eine Leerzeile einzufügen. Weitere eigene Eigenschaften besitzt das Tag nicht. Das End-Tag muss gesetzt werden.

So definieren Sie eigene Blocklevel-Tags:

1. `<div class="name">`
   `<div id="name">`
   `<div style="Definition">`
   Je nachdem, ob Sie eine Klasse, eine ID oder einen lokalen Stil anwenden möchten, setzen Sie im -Tag das entsprechende Attribut und den dazugehörigen Wert.
2. `Inhalt`
   Dann folgt der Inhalt des Abschnitts.
3. `</div>`
   Das End-Tag ist zwingend erforderlich.

Listing 2.19 zeigt den Quelltext einer Seite, in der mehrere -Tags eingesetzt wurden.

```
<html><head>
<title>Das div-Tag</title>
<style type="text/css"><!--
#blau {
font-family:Verdana,sans-serif;
font-size:13pt; color:#0000A0;
font-weight: bold }

#grau {
font-family:Verdana,sans-serif;
font-size:10pt; color:#404040;
text-align:justify;
margin-top:20px;
margin-bottom:20px }
-->
</style>
</head><body bgcolor="#FFFFFF">
<div id="blau">Blaue, fette Schrift </div>
<div id="grau">Graue Schrift als Fließtext. Der Abstand zu den Abschnitten vor und nach diesem wurde durch die CSS-Eigenschaften margin-top und margin-bottom erzeugt. Das div-Tag selbst beginnt nur eine neue Zeile, ohne einen Leerraum zwischen dem Block und den  umgebenden Elementen einzufügen. </div>
<div>Diesem div-Tag wurde kein Stil zugeordnet, also wird einfach nur vor und nach dem diesem Abschnitt eine neue Zeile begonnen. </div>Die nächste Textzeile. Sie steht hinter dem div-Abschnitt, eine neue Zeile wird begonnen.
</body></html>
```

Listing 2.19: Eigene Blocklevel-Tags

Abbildung 2.14: Das -Tag im Browser

Das -Tag in der Praxis

Das `<div>`-Tag wird Ihnen später in diesem Buch noch öfter begegnen. In DHTML spielt es eine wichtige Rolle, da mit `<div id="">` Objekte erzeugt werden, die mithilfe von JavaScript beeinflusst werden können und so Dynamik in die Webseite bringen.

Auch CSS-Layer werden mit dem `<div>`-Tag erzeugt. Layer sind Ebenen, die pixelgenau auf der Seite positioniert werden können und einander auch überlappen können. Mehr zu Ebenen erfahren Sie später.

In Abbildung 2.14 sehen Sie, wie der Browser den Quelltext von Listing 2.19 anzeigt.

Tag-spezifische Eigenschaften

Mit CSS können Sie die Formateigenschaften eines Tags ergänzen oder ändern. Solange Sie die ursprünglichen Eigenschaften eines Tags nicht gezielt ersetzen, bleiben diese erhalten. Nehmen wir als Beispiel das Tag <i>. Wir definieren für dieses Tag eine Stilregel:

```
i { color:#ff0000; }
```

Mit dieser Stilregel legen wir für das <i>-Tag die Schriftfarbe fest. Die ursprüngliche Eigenschaft des Tags bleibt erhalten, der Text wird auch weiterhin kursiv angezeigt.

Erst wenn wir diese Tag-spezifische Eigenschaft gezielt ändern, wird die Schrift nicht mehr kursiv angezeigt (siehe Listing 2.20 und Abbildung 2.15).

```
<html><head>
<title>Tag-Eigenschaften</title>
<style type="text/css">
<!--
i {
font-family:Verdana,sans-serif;
font-size:12pt;
color:#ff0000;
font-style:normal;
}
-->
</style>
</head><body bgcolor="#FFFFFF">
<p>Hier kommt <i>Text, der mit dem i-Tag
formatiert wurde.</i> <br>
Er wird aber nicht mehr kursiv angezeigt,
weil wir die Tag-spezifische Eigenschaft
überschrieben haben.</p>
</body></html>
```

Listing 2.20: Text mit dem <i>-Tag formatiert

Abbildung 2.15: Das Ergebnis von Listing 2.20 im Browser

```
<html><head><title>
Vererbte Tag-Eigenschaften
</title></head>
<body text="#0000a0"
bgcolor="#ffffff">
<p>Der Text in diesem Absatz wird blau
angezeigt. Diese Eigenschaft wird vom
Eltern-Tag body übernommen. <b><br>
Nun folgt eine Textpassage in Fett-Forma-
tierung, in der <i>kursiver Text</i> ein-
gebettet ist. </b> <br>
Der kursive Text ist zusätzlich fett, weil
das b-Tag als Eltern-Tag dem kursiven Text
die Eigenschaft fett vererbt.</p>
</body></html>
```

Listing 2.21: Vererbte HTML-Eigenschaften

Abbildung 2.16: Vererbte HTML-Eigenschaften im Browser

Vererbte Tag-Eigenschaften

Neben den Tag-spezifischen Eigenschaften gibt es auch vererbte Eigenschaften. Das gilt sowohl für HTML- als auch für CSS-Eigenschaften. Tags, die in anderen Tags eingebettet sind, übernehmen teilweise die Eigenschaften der äußeren Tags. Diese äußeren Tags nennt man auch Eltern-Tags; daher der Begriff „vererbt“. In vielen Publikationen wird der englische Begriff inherited verwendet, ich bevorzuge aber die deutsche Bezeichnung.

Vererbte HTML-Eigenschaften

Sehen wir uns zuerst einmal vererbte HTML-Eigenschaften an. Dazu verwenden wir das Beispiel in

1. Im `<body>`-Tag ist mit dem Attribut `text="0000a0"` die blaue Schriftfarbe festgelegt.
2. Das `<p>`-Tag ist im `<body>`-Tag eingebettet; die blaue Textfarbe wird vererbt.
3. Innerhalb des `<p>`-Tags ist das `<b>`-Tag eingebettet, darin wiederum das `<i>`-Tag. Beiden wird ebenfalls die blaue Schriftfarbe vererbt. Das `<i>`-Tag übernimmt zusätzlich die Fett-Formatierung seines Eltern-Tags `<b>`.

Das Ergebnis des Quellcodes sehen Sie in Abbildung 2.16.

Einige Eigenschaften können nicht vererbt werden. Wenn Sie z.B. im `<body>`-Tag Angaben für die Seitenränder machen, werden diese von den eingebetteten Tags nicht übernommen.

Vererbte CSS-Eigenschaften

Wie bei den HTML-Eigenschaften werden auch CSS-Eigenschaften vererbt, sofern das für die Eigenschaft möglich ist (siehe hierzu die CSS-Referenz im Anhang.)

Schauen wir uns wieder ein Beispiel an (siehe Listing 2.22):

1. Das <p>-Tag erbt die CSS-Eigenschaften, die für den Body festgelegt wurden.
2. Das <b>-Tag erbt die Eigenschaften seines Eltern-Tags <p> und fügt seine eigenen Merkmale (HTML und CSS) hinzu.
3. Das <i>-Tag erbt von seinem Eltern-Tag <b> ebenfalls alle Eigenschaften und fügt seine eigene Tag-spezifische Eigenschaft hinzu.

Wie der Quellcode aus Listing 2.22 im Browser angezeigt wird, sehen Sie in Abbildung 2.17.

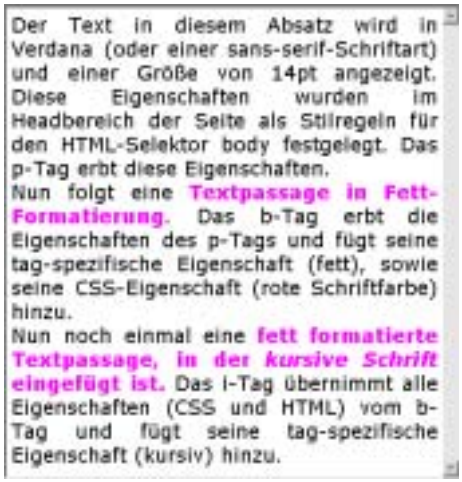

Abbildung 2.17: Vererbte CSS-Eigenschaften

```
<html><head><title>
Vererbte Tag-Eigenschaften
</title>
<style type="text/css">
<!--
body { font-size:12pt;
font-family:Verdana,sans-serif; }
b { color:#FF0000; }
-->
</style>
</head><body bgcolor="#ffffff">
<p>Der Text in diesem Absatz wird in Verdana (oder einer serifenlosen Schriftart) und einer Größe von 12pt angezeigt. Diese Eigenschaften wurden im Head-Bereich als Stilregeln für den HTML-Selektor body festgelegt. Das p-Tag erbt diese Eigenschaften. <br>
<b>Nun folgt eine Textpassage in Fett-Formatierung</b>. Das b-Tag erbt die Eigenschaften des p-Tags und fügt seine Tag-spezifische Eigenschaft (fett), sowie seine CSS-Eigenschaft (rote Schriftfarbe) hinzu. <br>
<b>Nun noch einmal eine fett formatierte Textpassage, in der <i>kursive Schrift</i> eingefügt ist.</b> Das i-Tag übernimmt alle Eigenschaften (CSS und HTML) vom b-Tag und fügt seine Tag-spezifische Eigenschaft (kursiv) hinzu.</p>
</body></html>
```

Listing 2.22: Vererbte Tag-Eigenschaften

```
body {
font-family:Verdana,sans-serif;
}

// Introbereich definieren:

#intro { …Stildefinitionen… }

/* Liste der Klassen und ihre Einsatz-
zwecke:
.navi1: Themen der Hauptnavigation
.navi2: Themen der Unternavigation
.fettblau: die ersten Zeilen einer
Tabelle, fette blaue Schrift
Ende des Kommentars */

.navi1 {
font-weight:bold;
font-size:12pt;
color:#000099;
}

.navi2 {
font-weight:bold;
font-size:10pt;
color:#000066;
}

.fettblau {
font-weight:bold;
font-size:12pt;
color:#0000ff;
```

Listing 2.23: Kommentare in CSS

Kommentare in CSS

Wenn Klassen oder IDs definiert werden, ist es von Vorteil, Kommentare in die Stilregeln einzufügen. So wissen Sie auch nach Wochen oder Monaten noch, welcher Stil für welchen Bereich der Seite zuständig ist. Solche Kommentare sind besonders dann zu empfehlen, wenn mehrere Personen an einem Projekt arbeiten.

Innerhalb von CSS können Sie nicht die Kommentarzeichen `<!--` und `-->` verwenden, die Sie von HTML kennen. Das würde dazu führen, dass die Browser die Stile nicht mehr ausführen.

Einigen Leserinnen und Lesern werden die CSS-Kommentarzeichen sehr vertraut sein, da sie auch in anderen Programmiersprachen, z.B. PHP, Anwendung finden.

Einzelne Kommentarzeilen:

```
// Hier folgt der Kommentar
```

Mehrzeilige Kommentare:

1. `/*`
   Geben Sie den Slash / und ein Sternchen * ein.
2. Schreiben Sie Ihre Kommentare. Dafür können Sie außer der Zeichenfolge `*/` alle beliebigen Zeichen, auch Sonderzeichen und die Returntaste, verwenden.
3. `*/`
   Beenden Sie den mehrzeiligen Kommentar durch die Eingabe der Zeichenfolge `*/`

Alles, was zwischen `/*` und `*/` steht, wird nun als Kommentar definiert.

Listing 2.23 zeigt Beispiele für Kommentare.

CSS für verschiedene Medien

Webseiten sind natürlich in erster Linie dazu gedacht, am Monitor angezeigt zu werden. Irgendwann wird aber bestimmt ein Besucher die Informationen Ihrer Webseite ausdrucken wollen.

Sie kennen das sicher aus eigener Erfahrung: Die Webseite hat einen dunklen Hintergrund mit heller Schrift oder eine Hintergrundgrafik und sieht im Browser wirklich toll aus. Aber druckerfreundlich ist eine solche Seite wahrhaftig nicht, denn der Tintenvorrat des Druckers wird arg beansprucht. Dabei waren Sie doch nur an den Textinformationen interessiert.

Häufig möchte man die ausgedruckte Seite auch in einem Sammelordner abheften, doch dafür lässt der normale Ausdruck nicht genug Rand an der linken Seite. Entweder schiebt man solche Seiten in Plastikhüllen, was auf die Dauer recht teuer werden kann, oder man nimmt in Kauf, dass der Locher einen Teil des Textes ausstanzt.

Mit CSS können sie Ihren Besuchern entgegenkommen. Sie haben die Möglichkeit, Ihre Seiten für den Druck zu optimieren.

Andere Medien

Theoretisch bieten CSS auch die Möglichkeit, die Webseiten für andere Medien zu optimieren, z.B. für Handys, Palmtops oder für Systeme, bei denen der Seiteninhalt über die Sprachausgabe erfolgt.

In der Praxis versteht aber bisher kein Gerät diese Befehle, daher wird in diesem Buch auch nicht darauf eingegangen.

```
body {
margin-left:2.5cm;
margin-top:2cm;
background:white;
font-family:times,serif;
font-size:12pt;
color:black;
}
…weitere Definitionen…
```

Listing 2.24: Beispiel für die optimierte Druckversion

✓ Tipps:

- Definieren Sie in der Druckversion einen breiten Rand, der ein problemloses Abheften ermöglicht.
- Schwarze Schrift auf weißem Untergrund schont den Farb-Tintentank des Druckers.
- Während am Monitor eine serifenlose Schrift wie Arial oder Verdana besser lesbar ist, gilt für Druckerzeugnisse das Gegenteil. In der Druckversion sollten Sie daher Serifen-Schriften wie Times einsetzen.

Seiten für den Druck optimieren

Optimierte CSS-Dateien

Sie benötigen zwei externe CSS-Dateien. Eine wird für die Anzeige am Bildschirm optimiert, die andere für den Druck. Wenn beide Versionen gleiche Stilregeln verwenden, sollten Sie eine dritte CSS-Datei anlegen, die dann alle Stile enthält, die für Monitor und Druckerversion gemeinsam verwendet werden.

Die verschiedenen zur Verfügung stehenden CSS-Eigenschaften lernen Sie in den nachfolgenden Kapiteln noch genau kennen. Listing 2.24 zeigt ein Fragment einer optimierten Druckversion.

Der Einsatz der optimierten Stile

Zum Einsatz der CSS-Dateien gibt es zwei unterschiedliche Methoden, die wir uns auf den nächsten Seiten näher anschauen werden.

Druckoptimierung: erste Methode

Bei der ersten Methode werden alle externen CSS-Dateien mit dem Link-Befehl in *einer* HTML-Seite eingebunden. Durch das Attribut `media` legen Sie fest, welches Gerät die jeweilige CSS-Datei verwendet. `media="screen"` ist dabei für die Ausgabe am Monitor zuständig, `media="print"` für den Ausdruck. Ein Beispiel sehen Sie in Listing 2.25.

```
<html><head>
<title>Methode 1</title>
<link rel="stylesheet" media="print"
href="druck.css" type="text/css">
<link rel="stylesheet" media="screen"
href="monitor.css" type="text/css">
</head>
<body>
Seiteninhalte
</body></html>
```

Listing 2.25: Beide CSS-Dateien werden in einer HTML-Seite eingebunden.

So werden die optimierten CSS-Dateien eingebunden:

1. `<link rel="stylesheet" media="print" href="druck.css" type="text/css">`
   Setzen Sie den Cursor in den Head-Bereich Ihrer Seite und geben Sie den Link-Befehl zum Einfügen der Druckversion ein. Ersetzen Sie die Angabe `druck.css` durch den Dateinamen, den Sie für die Druckversion gewählt haben.
2. `<link rel="stylesheet" media="screen" href="screen.css" type="text/css">`
   Setzen Sie den Cursor in den Head-Bereich Ihrer Seite und geben Sie den Link-Befehl zum Einfügen der Monitorversion ein. Ersetzen Sie die Angabe `screen.css` durch den Dateinamen, den Sie für die Monitorversion gewählt haben.

Diese Methode hat leider einen entscheidenden Nachteil: Nur der IE5 kommt damit richtig zurecht. Manche Browser ignorieren diese CSS-Angaben ganz, andere finden zwar die richtige CSS-Datei für die Bildschirmausgabe, ignorieren jedoch die Druckformate.

✓ Tipp:

- Die Reihenfolge, in der die CSS-Dateien eingebunden werden, spielt eine Rolle, denn Browser, die den `media`-Befehl nicht verstehen, verwenden ggf. beide CSS-Dateien. Da in beiden Dateien zumindest teilweise unterschiedliche Definitionen für einen Bereich (z.B. Body) vorhanden sind, kommt es dabei zu Konflikten. Die Browser verwenden die Stile, die zuletzt angesprochen werden. Daher sollte die CSS-Datei für die Monitoranzeige immer als letzte eingebunden werden.

✓ Tipps:

- Mit dieser Methode können Sie Ihren Besuchern sehr viel Komfort bieten, wenn Sie etwas mehr Arbeit investieren und die Druckversion noch überarbeiten:
- Die Besucher sind vor allem an den Textinformationen interessiert. Löschen Sie also alle Grafiken, die nicht unbedingt erforderlich sind. Navigationsbuttons kann man z.B. gegen Textlinks austauschen.
- Wenn Sie Anleitungen oder Workshops auf Ihren Seiten anbieten, für die Grafiken oder Illustrationen unbedingt erforderlich sind, prüfen Sie bitte, ob es ausreicht, die Grafiken für den Ausdruck in einer Graustufenversion anzubieten. Das ist fast immer der Fall und gute Bildbearbeitungsprogramme bieten die Möglichkeit zur Stapelverarbeitung. So wandeln Sie die farbigen Illustrationen im Nu in Graustufen um.
- Wenn viel JavaScript oder DHTML auf den Seiten eingesetzt wird, sollte man für Besucher mit älteren Browsern immer eine "abgespeckte" Version anbieten. In der Druckversion benötigen Sie die Scripts und DHTML ebenfalls nicht. Verwenden Sie hier einfach die gleiche Datei, so haben Sie zwei Fliegen mit einer Klappe geschlagen.

Druckoptimierung: zweite Methode

Da die erste Methode viele Browser ausschließt, bevorzuge ich persönlich die zweite Methode. Sie ist mit mehr Aufwand verbunden und benötigt auch mehr Webspace auf dem FTP-Server. Da dieser aber sehr kostengünstig zu haben ist, sollte das nicht das entscheidende Kriterium sein. Den höheren Arbeitsaufwand sollte man für zufriedene Besucher gerne in Kauf nehmen.

Bei der zweiten Methode wird die HTML-Seite in zwei Versionen gespeichert: einmal ganz normal zur Anzeige am Monitor und einmal als Druckversion.

In die Druckversion wird die für den Druck optimierte CSS-Datei eingebunden. In die Monitorversion binden Sie die normale CSS-Datei ein und setzen einen Link auf die Seite, die für den Druck optimiert ist.

Dieses Verfahren mag auf den ersten Blick sehr aufwändig erscheinen, es geht aber schneller, als man denkt. Schreiben Sie zuerst die Monitorversion der Seite und speichern Sie die fertige Datei dann unter einem anderen Namen. Hier bietet sich an, dem Dateinamen einfach `druck_` voranzustellen. In dieser Datei tauschen Sie dann nur noch den Namen der externen CSS-Datei aus. In der Monitorversion setzen Sie den Link auf die Druckversion – fertig!

3

SCHRIFTEN

Das Grundelement eines Textes ist die Schrift. Das Aussehen eines Textes wird im Wesentlichen von zwei Faktoren bestimmt: der verwendeten Schrift und der Formatierung des Textabschnitts selbst.

Reines HTML lässt Ihnen wenig Möglichkeiten, Schriften optisch aufzubereiten. In der aktuellen HTML4-Spezifikation wurden übrigens etliche Tags vom W3C missbilligt. Solche Tags gehören zwar noch zum Standard, werden aber in zukünftigen Versionen ausgeschlossen. Zu diesen Tags gehören auch Befehle für die Schriftformatierung, z.B. `<font>`, der in HTML derzeit die einzige Möglichkeit bietet, eine Schriftart, -größe oder -farbe festzulegen.

Für Schriften lautet die eindeutige Empfehlung des W3C: Benutzen Sie CSS.

CSS erlauben Ihnen die Kontrolle über das Erscheinungsbild der Schriften. Verschiedene Textfarben, Schriftgrößen und Auszeichnungen – alles kein Problem mit CSS.

Schriften auf Webseiten

Beim Einsatz von Schriften auf Webseiten gibt es Einiges, was Sie beachten sollten. Theoretisch können Sie jede Schrift auf Ihrer Seite verwenden. Doch was passiert, wenn auf dem Rechner des Besuchers diese Schrift nicht installiert ist? Ganz einfach: Der Browser verwendet seine Standardschriftart, die Webseite sieht dann ggf. ganz anders aus als auf Ihrem Computer.

Welche Schriften auf einem Rechner zur Verfügung stehen, ist abhängig vom Betriebssystem und der installierten Browsersoftware. Als Webdesigner können Sie natürlich nicht wissen, welche Voraussetzungen beim Besucher gegeben sind, also muss man nach Kompromissen suchen.

Ansatzpunkt hierfür ist der kleinste gemeinsame Nenner. Bei Schriften sind das die Schrifttypen, die auf jedem Rechner vorhanden sind: *Serif*, *Sans-serif* und *Monospace*. Für diese generischen Schrifttypen hat jedes Betriebssystem eine installierte Systemschriftart. Einige Systeme kennen auch die Typen *Cursive* und *Fantasy*, sie sind für Fließtexte aber nicht geeignet.

Bevor wir dazu kommen, wie man auf dem gemeinsamen Nenner aufbaut, sehen wir uns die unterschiedlichen Schrifttypen an.

Downloadbare Schriften

Es gibt die Möglichkeit, eine Schriftart so in die Seite einzubetten, dass die Schrift vom Server heruntergeladen wird. In der Praxis erweist sich diese Technik aber leider immer noch als unbrauchbar.

Außer dem Internet Explorer und Netscape kann kein Browser diese Schriften verwenden. Selbst für diese beiden Browser müssen unterschiedliche Techniken eingesetzt werden, zum Erstellen der downloadbaren Schriften sind unterschiedliche Programme erforderlich. Zudem gibt es auch Probleme mit dem Copyright der Schriften.

Hohe Kosten und Aufwand stehen derzeit einem sehr geringen Nutzen gegenüber. Aus diesem Grund wird in diesem Buch auf downloadbare Schriften nicht näher eingegangen.

Abbildung 3.1: Oben sehen Sie den Schriftzug in der Serifenschrift Times, darunter in der serifenlosen Schriftart Arial.

Abbildung 3.2: Ein Schriftzug in der Monospace-Schriftart Courier.

Schrifttypen

- *Serif:* Die Serifenschriften zeigen typische abgerundete Abschlussstriche an den Buchstaben (siehe Abbildung 3.1). In Printerzeugnissen sind Serifenschriften die erste Wahl. Am Monitor und damit auch bei Webseiten sollten Serifenschriften bei Fließtexten vorsichtig eingesetzt werden, da die Serifen bei kleinen Schriftgrößen das Schriftbild leicht verschwimmen lassen. Für Überschriften sind sie aber gut geeignet. Ein typischer Vertreter der Serifenschriften ist Times.
- *Sans-serif: sans* ist das französische Wort für *ohne*. Genau das beschreibt den Charakter dieser Schriften: Ihnen fehlen die Serifen (siehe Abbildung 3.1). Serifenlose Schriften haben ein klareres Schriftbild und sind für Fließtexte in Webseiten daher besser geeignet als Serifenschriften. Arial ist eine serifenlose Schrift.
- *Monospace:* Das besondere Merkmal dieser Schriften ist die Breite der Buchstaben (siehe Abbildung 3.2). Bei Proportional-Schriften wie Times und Arial ist die Breite der Buchstaben variabel, der Buchstabe i braucht weniger Platz als der Buchstabe m. Bei den Monospace-Schriften ist das anders: Hier benötigen alle Buchstaben gleich viel Platz. Eine typische Monospace-Schriftart ist Courier.

Auf die Schrifttypen *Cursive* und *Fantasy* wird in diesem Buch nicht eingegangen, da sie in der Praxis nur sehr bedingt einsetzbar sind. Diese Schrifttypen kennen nur einige Systeme. Hier ist das Risiko, dass nur wenige Besucher das gewünschte Ergebnis wirklich sehen, einfach zu groß.

Schriftarten und Schriftfamilien

Schriftarten mit gleichem Aussehen haben auf den verschiedenen Betriebssystemen oft auch unterschiedliche Namen. So heißt z.B. die Windows-Schriftart Arial auf dem Macintosh Helvetica. Einander ähnliche Schriftarten von Windows und Mac sehen Sie in Abbildung 3.3.

Windows	Mac	Typ
Arial	Helvetica	Sans-serif
Times New Roman	Times	Serif
Courier New	Courier	Monospace

Abbildung 3.3: Die Systemschriften im Vergleich

HTML und natürlich auch CSS erlauben den Einsatz von Schriftfamilien. In einer solchen Schriftfamilie werden mehrere Schriftarten hintereinander notiert. Der Browser prüft, ob die erste angegebene Schriftart auf dem System installiert ist. Ist das der Fall, wird der Text in dieser Schriftart angezeigt. Wenn es die angegebene Schriftart auf dem System nicht gibt, wird die zweite Angabe geprüft. So arbeitet sich der Browser durch die ganze Schriftliste. Erst wenn keine der angegebenen Schriftarten gefunden wird, verwendet der Browser seine Standardschriftart.

Die Schriftfamilien bieten die Möglichkeit, ähnliche Schriftarten für verschiedene Betriebssysteme zu notieren. Als erste Angabe können Sie ihre bevorzugte Schriftart notieren. Schmuckschriften sollten Sie dabei aber vermeiden. Als letzte Angabe sollte immer der generische Schrifttyp stehen.

Die gebräuchlichen Schriftfamilien finden Sie in Abbildung 3.4. Wie man Schriftfamilien mit CSS setzt, folgt später in diesem Kapitel.

Schriftfamilien:

Arial, Helvetica, sans-serif

Verdana, Arial, Helvetica, sans-serif

Times New Roman, Times, serif

Georgia, Times New Roman, Times, serif

Courier New, Courier, monospace

Abbildung 3.4: Browseransicht der gebräuchlichen Schriftfamilien

✓ Tipp:

- Setzen Sie in der Schriftfamilie immer den entsprechenden Schrifttyp (*serif, sans-serif, monospace*) an die letzte Stelle. So sehen die Besucher zumindest eine Schriftart, die der von Ihnen gewünschten ähnlich sieht.

Abbildung 3.5: Grafische Elemente für Überschriften und Initialen

Überschriften und Initialen

CSS bieten Ihnen viel mehr Möglichkeiten als HTML, um Überschriften und Initialen zu realisieren. Häufig möchte man aber für solche Elemente eine besondere Schmuckschrift einsetzen, die dann auch auf allen Besucherrechnern angezeigt werden soll.

In solchen Fällen kann man sich mit Grafiken behelfen. Der gewünschte Schriftzug wird in einem Bildbearbeitungsprogramm erstellt und als *.gif* oder *.jpg* abgespeichert. Diese Grafik wird dann im Text integriert (siehe Abbildung 3.5).

Kommen Sie nun bitte nicht auf die Idee, ganze Textpassagen auf diese Weise zu erstellen. Grafiken bedeuten immer höhere Ladezeiten; außerdem würden Besucher von dem „Text“ nichts zu sehen bekommen, wenn die Grafikanzeige im Browser abgeschaltet ist. Verwenden Sie solche Elemente sparsam.

Die Schriftfamilie festlegen

Die Eigenschaft zum Festlegen einer Schriftfamilie lautet `font-family`. Das trifft auch zu, wenn Sie statt einer Familie nur eine Schriftart angeben (was Sie aber vermeiden sollten).

So legen Sie die Schriftfamilie fest:

1. `font-family:`
   Geben Sie die Eigenschaft `font-family` ein, gefolgt von einem Doppelpunkt.
2. `Schriftart1,Schriftart2,`
   Geben Sie nacheinander die Schriftarten an. Hinter jeder Schriftart wird ein Komma gesetzt, Leerzeichen sind nicht erforderlich.
3. `Schrifttyp;`
   An letzter Stelle steht der Schrifttyp (z.B. serif), gefolgt von dem Semikolon, der die Stilregel abschließt.

Schriftarten, die aus mehreren Wörtern bestehen, z.B. `Times New Roman`, sollen nach Empfehlung des W3C in Anführungszeichen gesetzt werden. In lokalen Stilen müssen Sie dabei einfache Anführungszeichen verwenden, da die Schriftfamilie der Wert des `style`-Attributs ist und in doppelten Anführungszeichen steht.

In Listing 3.1 sehen Sie Schriftfamilien im Einsatz als interne und lokale Stile. Abbildung 3.6 zeigt das Ergebnis im Browser.

```
<html><head>
<title>Schriftfamilien</title>
<style type="text/css">
<!--
p { font-family:Georgia,Times,
"Times New Roman",serif; }
-->
</style>
</head><body text="#000000"
bgcolor="#FFFFFF">
<p>Ein Absatz in Serifschrift. Der Stil
wurde als interne Stilregel für den HTML-
Selektor p festgelegt. </p>
<p style="font-family:Courier,
'Courier New',monospace;">Dieser Absatz
wurde mit einem lokalen Stil in Monospace-
Schrift formatiert. </p>
</body></html>
```

Listing 3.1: Schriftfamilien festlegen...

Abbildung 3.6: ...und die Anzeige im Browser

```
<html><head>
<title>Schriftfamilien</title>
<style type="text/css">
<!--
.groesse1 { font-size:0.3in; }
.groesse2 { font-size:12pt; }
.groesse3 { font-size:12px; }
.groesse4 { font-size:larger; }
-->
</style>
</head><body text="#000000"
bgcolor="#FFFFFF">
<p>Hier sehen Sie verschiedene Schriftgrö-
ßen, die mit span-Tags gesetzt wurden:
</p>
<p> <span class="groesse1">0.3in
</span> <span class="groesse2">12pt
</span> <span class="groesse3">12px
</span> <span class="groesse4">
larger</span></p>
</body></html>
```

Listing 3.2: Schriftgrößen...

Abbildung 3.7: ... und Darstellung im Browser

Schriftgrößen festlegen

HTML-Schriftgrößen sind immer relative Größen, also abhängig von den Eigenschaften des Besucherrechners. CSS bieten neben relativen Schriftgrößen auch die Möglichkeit, feste Größen zu definieren.

So legen Sie die Schriftgröße fest:

1. `font-size:`
   Geben Sie die Eigenschaft `font-size` ein, gefolgt von einem Doppelpunkt.
2. `Schriftgröße;`
   Setzen Sie dahinter die gewünschte Größe, gefolgt von einem Semikolon, das die Stilregel abschließt.

Mit dem `<span>`-Tag können Sie jederzeit verschiedene Schriftgrößen im Fließtext setzen. Ein Beispiel finden Sie in Listing 3.2, das Ergebnis sehen Sie in Abbildung 3.7.

Erlaubte Werte für Schriftgrößen

Als Schriftgröße können Sie nicht nur alle numerischen Werte verwenden, sondern auch beschreibende Werte wie `small` oder `smaller`. Die numerischen Angaben wurden bereits im Abschnitt *Angaben für die CSS-Werte* in Kapitel 1 besprochen.

Die erlaubten beschreibenden Werte geordnet von der kleinsten bis zur größten Angabe lauten:

`xx-small, x-small, small, medium, large, x-large, xx-large`

Außerdem gibt es noch die relativen Angaben `smaller` und `larger`.

Wie beschreibende Werte im Browser aussehen, zeigt die Abbildung 3.8.

Die relativen Werte `smaller`, `larger` sowie Prozentangaben beziehen sich immer auf die Schriftgröße des übergeordneten Stils. Wenn Sie z.B. in einem `<p>`-Tag, für das die Schriftgröße 12pt gesetzt ist, Text mit einer Schriftgröße von 150% formatieren, ist dieser Text 18pt groß (siehe Abbildung 3.9). Ohne einen übergeordneten Stil bezieht sich eine solche Angabe auf die Standardschriftgröße des Browsers.

Abbildung 3.8: Die beschreibenden Werte für Schriftgrößen

Abbildung 3.9: Prozentuale Angaben beziehen sich auf den übergeordneten Stil.

✓ Tipp:

- Wie groß die Schrift angezeigt wird, ist auch abhängig vom Betriebssystem und der Browsersoftware. Wenn Sie für Schriftgrößen Pixel- oder Punktangaben verwenden, wird die Schrift auf möglichst vielen Rechnern gleich groß angezeigt.

Abbildung 3.10: Ansicht der Stile `italic` und `oblique` im Opera und IE

Abbildung 3.11: Die gleiche Seite im Netscape 4. Er zeigt den `oblique`-Stil nicht an.

Kursive Schrift

`font-style` steht für den Schriftstil. Damit ist die Neigung der Schrift gemeint. CSS kennt drei Angaben: `normal`, `italic` und `oblique`.

Die beiden Stile `italic` und `oblique` führen oft zu Verwirrung, da beide die Schrift kursiv darstellen. Der Unterschied liegt bei den verwendeten Schriftarten. Einige haben einen integrierten kursiven Stil, der angezeigt wird, wenn man den Stil `italic` verwendet. Beim Einsatz von `oblique` wird die Schrift einfach leicht nach rechts geneigt.

Zumindest sollte das so sein, in der Praxis machen die Browser aber keine Unterschiede. Schriften, die den integrierten Kursivstil haben, werden bei beiden Angaben `italic` angezeigt, die anderen immer `oblique`. Sie können die `oblique`-Angabe getrost vergessen, vor allem, da der Netscape-Browser sie auf Grund eines Bugs nicht anzeigt. Verwenden Sie stattdessen immer die Angabe `italic`.

In Abbildung 3.10 sehen Sie die beiden Schriftstile im direkten Vergleich, wie sie im IE und Opera angezeigt werden. Netscape ignoriert die `oblique`-Angabe (siehe Abbildung 3.11).

So wenden Sie einen Schriftstil an:

1. `font-style:`
   Geben Sie die Eigenschaft `font-style` ein, gefolgt von einem Doppelpunkt.
2. `Schriftstil;`
   Schreiben Sie dahinter den gewünschten Schriftstil, gefolgt von dem Semikolon, der die Stilregel abschließt.

Mit dem Stil `normal` können Sie in Textblöcken, die komplett kursiv gesetzt sind, für bestimmte Textpassagen den Schriftstil wieder auf `normal` setzen. Ein Beispiel dafür sehen Sie in Listing 3.3. Abbildung 3.12 zeigt das Ergebnis im Browser.

```
<html><head>
<title>Schriftstile</title>
<style type="text/css">
<!--
p { font-family:Arial,sans-serif;
font-size:12pt; font-style:italic;
text-align:justify; }
h2 { font-size:20pt;
font-family:Georgia,Times,serif; }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
<h2>Die Highlands</h2>
<p>Die Burgen in den
<span style="font-style:normal;">
schottischen Highlands</span> sind häufig
sehr gut erhalten. In einigen der alten
Gemäuer können Sie ein Zimmer mieten, um
eine Nacht in stilvoller Atmosphäre zu
verbringen.</p>
</body></html>
```

Listing 3.3: Schriftsstil festlegen...

Abbildung 3.12: ...und die Anzeige im Browser

```
<html><head>
<title>Fettformate</title>
<style type="text/css">
<!--
.fett {
font-family:Arial,sans-serif;
font-size:12pt; font-weight:700; }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
<p><span class="fett">Fett formatierter
Text und <span style=" font-weight:nor-
mal;">mittendrin normaler </span> Text.
</span> </p>
</body></html>
```

Listing 3.4: Schriften gewichten

Fett formatierter Text und mittendrin normaler Text.

Abbildung 3.13: Ergebnis fett und wieder normal

Abbildung 3.14: Anzeige im Browser

Fette Schrift

Die Dicke einer Schrift nennt man auch Schriftgewicht. Die Eigenschaft dafür heißt `font-weight`. HTML kennt nur die beiden Angaben fett oder nicht fett, in CSS kann man etliche unterschiedliche Schriftgewichte definieren. Allerdings unterstützt keine Schriftart alle Werte.

Erlaubte Werte sind `normal`, `bold`, `lighter`, `bolder` sowie volle Hunderterschritte zwischen `100` und `900`. Bei den numerischen Werten entspricht `100` der dünnsten Schrift und `900` der fettesten. `500` steht für normales Schriftgewicht, `700` entspricht der Angabe `bold`.

Die beiden Angaben `lighter` (dünner) und `bolder` (fetter) beziehen sich auf den umgebenden Text. Mit diesen Angaben haben viele Browser Probleme, u.a. auch Netscape 4.

So wenden Sie Schriftgewichte an:

1. `font-weight:`
   Geben Sie die Eigenschaft `font-weight` ein, gefolgt von einem Doppelpunkt.
2. `Schriftgewicht;`
   Schreiben Sie dahinter den gewünschten Fettigkeitsgrad, gefolgt von einem Semikolon, das die Stilregel abschließt.

Mit dem Stil `normal` können Sie z.B. in Tags, für die zentral eine Fettformatierung festgelegt ist, das Schriftgewicht wieder auf `normal` setzen. Ein Beispiel sehen Sie in Listing 3.4 und das Ergebnis in Abbildung 3.13.

Weitere Beispiele finden Sie in Abbildung 3.14.

Kapitälchen

Für Überschriften sind die Kaptitälchen ein wirkungsvolles Stilmittel; die CSS-Eigenschaft hierfür lautet `font-variant`.

So setzen Sie Kapitälchen:

1. `font-variant:`
   Geben Sie die Eigenschaft `font-variant` ein, gefolgt von einem Doppelpunkt.
2. `small-caps;`
   Schreiben Sie dahinter den Wert `small-caps`, gefolgt von einem Semikolon, das die Stilregel abschließt.

Außer dem Wert `small-caps` ist auch `normal` als Angabe erlaubt. Damit können Sie in Textpassagen, die mit `small-caps` formatiert sind, für bestimmte Textstellen das Schriftformat wieder auf normal setzen.

In Listing 3.5 sehen Sie ein Beispiel für Kapitälchen. Viele Browser haben leider noch Probleme mit den Kapitälchen. Nur Opera zeigt sie so an, wie es sein soll: Wenn der erste Buchstabe eines Wortes wirklich groß geschrieben ist, wird er etwas größer und fetter dargestellt als die restlichen Kapitälchen (siehe Abbildung 3.15). IE5.x setzt zwar Großbuchstaben, ignoriert aber die Hervorhebung der groß geschriebenen Wortanfänge (siehe Abbildung 3.16). Netscape 4.x definiert den Befehl gar nicht (siehe Abbildung 3.17).

```
<html><head>
<title>Fettformate</title>
<style type="text/css">
<!--
.kapi { font-variant:small-caps;
font-size:14pt; }
p { font-family:Verdana,sans-serif;
font-size:12pt; }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
<p class="kapi">Die schottischen
Highlands</p>
<p>Burgen sind in den Highlands...
</p></body></html>
```

Listing 3.5: Fettformate

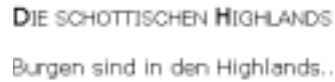

Abbildung 3.15: Die Anzeige in Opera ist korrekt.

Abbildung 3.16: IE5.x zeigt immerhin noch Großbuchstaben an.

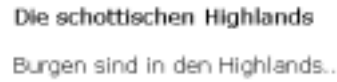

Abbildung 3.17: Netscape4.x ignoriert die Kapitälchen.

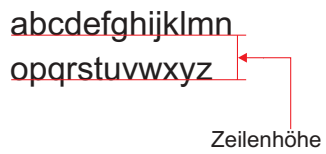

Abbildung 3.18: Zeilenhöhe

Zeilenhöhen einrichten

Im DTP-Bereich sind Zeilenhöhen ein gebräuchliches Mittel, um Fließtext besser lesbar zu machen. Im Fachjargon werden die Zeilenhöhen als „Durchschuss“ (engl. leading) bezeichnet. Diesen Begriff findet man auch in vielen Textverarbeitungsprogrammen wieder.

Die Zeilenhöhe bezeichnet den Zwischenraum zwischen den Grundlinien der Schrift (siehe Abbildung 3.18). Sie steht in unmittelbarem Zusammenhang mit der Schriftgröße.

Die Zeilenhöhe sollte für Fließtext nicht zu klein, aber auch nicht zu groß gewählt werden. Im DTP-Bereich gilt als Faustregel ein Wert von `120%`, das bedeutet, dass bei einer 10 Punkt großen Schrift der Zeilenabstand 12 Punkt beträgt.

Natürlich müssen Sie sich nicht an diese Regel halten, sie ist aber ein guter Anhaltspunkt. Für Effekte wählen Sie größere oder kleinere Werte. Mehrzeilige Überschriften sehen meist schöner aus, wenn für die Zeilenhöhe ein Wert von `95` oder `100%` gewählt wird.

✓ Tipp:

- Viele Browser verwenden für die Zeilenhöhe den DTP-Standard `120%`.

So legen Sie die Zeilenhöhe fest:

1. `line-height:`
   Geben Sie die Eigenschaft `line-height` ein, gefolgt von einem Doppelpunkt.
2. `Zeilenhöhe;`
   Schreiben Sie dahinter den Wert für die Zeilenhöhe, gefolgt von einem Semikolon, das die Stilregel abschließt.

Erlaubte Werte für die Zeilenhöhe sind alle numerischen Angaben:

- Zahlen ohne Einheiten als Multiplikator der Schriftgröße, z.B. `1.5`.
- Prozentwerte beziehen sich ebenfalls auf die Schriftgröße.
- Die Einheit, die für die Schriftgröße verwendet wird. Bei einer Schriftgröße mit `10pt` wird also auch für die Zeilenhöhe ein `pt`-Wert gewählt.

Listing 3.6 zeigt ein Beispiel, in Abbildung 3.19 sehen Sie das Ergebnis im Browser.

```
<html><head>
<title>Zeilenhöhen</title>
<style type="text/css">
<!--
p { line-height:1.3;
font-family:Verdana,sans-serif;
font-size:11pt; }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
<p>Burgen und Schlösser sind in Schottland
häufig noch gut erhalten. In einigen der
alten Gemäuer können Sie ein Zimmer mie-
ten, um eine Nacht in stilvoller Atmo-
sphäre zu verbringen.</p>
<p style="line-height:160%">Im Hochland
sind die Burgen wehrhafter als im Süden
des Landes. Die dicken Mauern trotzen seit
Jahrhunderten den widrigen Wetterverhält-
nissen und schützen ihre Bewohner vor den
kalten Nordstürmen.</p>
</body></html>
```

Listing 3.6: Zeilenhöhe festlegen

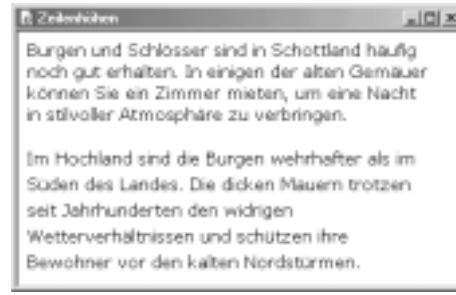

Abbildung 3.19: Zeilenhöhe im Browser

```
.alles {
font-family:Arial,Helvetica,sans-serif;
font-size:12pt;
line-height:14pt;
font-style:italic;
font-weight:bold;
font-variant:small-caps;
}
```

Listing 3.7: Kurzschrift für Schriftwerte

✓ Tipp:

- Die Kurzschrift mit `font` spart viel Schreibarbeit, außerdem ist der Quelltext bedeutend kürzer. Dadurch bleibt der Code nicht nur übersichtlicher, auch die Ladezeiten sind geringer.

Schriftwerte in Kurzschrift

Nun haben Sie die Möglichkeiten kennen gelernt, das Erscheinungsbild der Schrift direkt zu beeinflussen. Wenn Sie alle diese Eigenschaften verwenden wollen, haben Sie jede Menge zu tippen (siehe Listing 3.7).

Wäre es nicht angenehmer, die Schrifteigenschaften in kürzerer Form schreiben zu können? CSS bieten Ihnen diese Möglichkeit! Sie können die Schrifteigenschaften in einer Kurzschrift zusammenfassen; der Befehl hierfür lautet schlicht und einfach `font`.

Folgende Schrifteigenschaften lassen sich mit `font` zusammenfassen:

- `font-style`
- `font-variant`
- `font-weight`
- `font-size`
- `line-height`
- `font-family`

Für die Kurzschrift gelten bestimmte Regeln, die man einhalten muss, siehe den *Infokasten Strikte Regeln*. Davon sollte man sich aber nicht abschrecken lassen. Die Regeln sind zwar strikt, sind aber leicht zu merken.

So wenden Sie die Kurzschrift für die Schrifteigenschaften an:

1. `font:`
   Geben Sie die Eigenschaft `font` ein, gefolgt von einem Doppelpunkt.
2. `italic`
   Wenn Sie Kursivschrift wünschen, geben Sie `italic` ein, gefolgt von einem Leerzeichen. Wenn die Schrift nicht kursiv dargestellt werden soll, können Sie den Schriftstil weglassen.
3. `small-caps`
   Für Kapitälchen geben Sie `small-caps` ein, gefolgt von einem Leerzeichen. Wenn Sie die Eigenschaft nicht benötigen, können Sie sie weglassen.
4. `bold`
   Für das Schriftgewicht geben Sie einen erlaubten Wert ein, gefolgt von einem Leerzeichen. Wenn Sie die Eigenschaft nicht benötigen, können Sie sie weglassen.
5. `Schriftgröße`
   Geben Sie einen erlaubten Wert für die Schriftgröße ein. *Die Schriftgröße darf nicht weggelassen werden*! Wenn Sie nachfolgend keine Zeilenhöhe verwenden möchten, geben Sie nach der Schriftgröße ein Leerzeichen ein.
6. `/Zeilenhöhe`
   Fügen Sie einen Slash / und direkt dahinter einen erlaubten Wert für die Zeilenhöhe ein, gefolgt von einem Leerzeichen. Die Zeilenhöhe kann weggelassen werden.

Fortsetzung auf der nächsten Seite

Strikte Regeln

Wenn Sie die Langform der Schrifteigenschaften verwenden, ist es egal, in welcher Reihenfolge die Merkmale stehen.

Für die Kurzschrift mit `font` wurde – aus welchen Gründen auch immer – eine Reihenfolge festgelegt. Wenn sie missachtet wird, reagieren die Browser unterschiedlich darauf. Die eine oder andere Eigenschaft wird angezeigt, aber in keinem Fall sieht die Schrift so aus wie in der richtigen Reihenfolge!

Halten Sie sich daher immer an die richtige Reihenfolge. Sie lautet: `font-style, font-variant, font-weight, font-size, line-height, font-family.`

Wenn Sie eine Eigenschaft nicht verwenden wollen, lassen Sie diese weg. `font-size` *und* `font-family` *müssen aber in jedem Fall gesetzt werden, sonst funktioniert die Kurzschrift-Stilregel nicht!* Bei den Eigenschaften `font-style`, `font-variant` und `font-weight` kann man auch den Wert normal setzen.

Die Eigenschaft `line-height` kann nur in direkter Verbindung mit der Schriftgröße gesetzt werden. Die Zeilenhöhe wird mit einem Slash / an die Schriftgröße angehängt.

```
<html><head>
<title>Kurzschrift</title>
<style type="text/css"><!--
.eins { font:normal normal bold 12pt/120%
Arial,sans-serif; }
.zwei { font:bold 12pt/120% Arial,sans-
serif; }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
<p class="eins">Die Eigenschaften für die-
sen Absatz wurden in Kurzschrift zusammen-
gefasst.<br>Die Eigenschaften <i>font-
style</i> und <i>font-variant</i> wurden
nicht benötigt, die Werte sind mit <i>nor-
mal</i> angegeben, um die Formatierung zu
unterdrücken. </p>
<p class="zwei">Dieser Absatz hat die
gleichen Eigenschaften wie der erste
Absatz. In der Kurzschrift wurden diesmal
die nicht benötigten Eigenschaften wegge-
lassen, statt sie mit dem Wert <i>normal
</i> anzugeben. </p>
</body></html>
```

Listing 3.8: Kurzschrift mit font

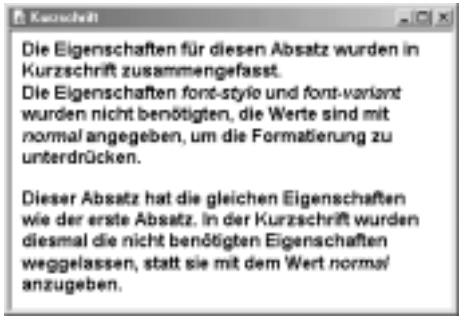

Abbildung 3.20: Aussehen im Browser

7. `Arial,Helvetica,sans-serif;`
Schreiben Sie die gewünschten Schriftarten auf, getrennt durch Kommas. Am Ende der Schriftliste steht wie üblich der generische Schrifttyp, gefolgt von einem Semikolon, das die Kurzschrift-Stilregel abschließt. *Die Schriftfamilie darf nicht weggelassen werden*!

Wenn Sie die Eigenschaften `font-style`, `font-variant` und `font-weight` nicht benötigen, können Sie diese Eigenschaften weglassen oder den Wert normal setzen.

Listing 3.8 zeigt ein Beispiel für die Kurzschrift mit `font`, in Abbildung 3.20 sehen Sie das Ergebnis im Browser.

Schriftfarben bestimmen

Verschiedene Textfarben sorgen für die Hervorhebung einzelner Buchstaben, Wörter oder ganzer Textpassagen. Sie können auch zur optischen Gliederung von Texten beitragen. Die Eigenschaft zum Festlegen der Textfarbe heißt `color`.

So legen Sie eine Schriftfarbe fest:

1. `color:`
   Geben Sie die Eigenschaft `color` ein, gefolgt von einem Doppelpunkt.
2. `#ff0000;`
   Schreiben Sie dahinter den gewünschten Farbwert, gefolgt von dem Semikolon, der die Stilregel abschließt.

Die Farbwerte können Sie mit verschiedenen Schreibweisen angeben (siehe auch den Abschnitt *Angaben für die CSS-Werte* in Kapitel 1). Weit verbreitet ist die oben gezeigte Methode mit Hexadezimalwerten. Bei dieser Schreibweise ist das Rautezeichen ein zwingend erforderlicher Bestandteil des Farbwertes.

Einzelne Buchstaben oder Wörter können Sie mit dem `<span>`-Tag farbig hervorheben. Listing 3.9 zeigt ein Beispiel für farbige Überschriften sowie den lokalen Einsatz; in Abbildung 3.21 sehen Sie das Ergebnis im Browser.

```
<html><head>
<title>Farbige Schrift</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif; }
h2 { font:18pt Georgia,serif;
color:#e00000; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
<h2>Gliederung durch Farben </h2>
<p>Farbige Überschriften sorgen bei
längeren Texten für die optische
<span style="color:#e00000;"> Gliederung
</span>der Themen.</p>
</body></html>
```

Listing 3.9: Farbige Schrift…

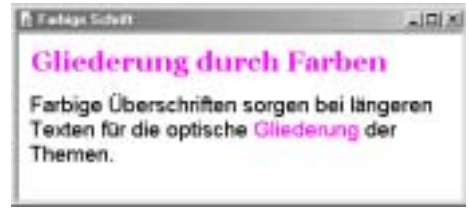

Abbildung 3.21: …im Browser

✓ Tipp:

- Manche Browser benutzen `color` als allgemeine Vordergrundfarbe. Rahmen werden dann z.B. auch in dieser Farbe angezeigt.
- Die verwendeten Farben sollten ein stimmiges Erscheinungsbild ergeben und müssen immer auch auf das Ziel der Seite abgestimmt werden. Eine Webseite für Kinder verträgt mehr Farbe als eine Firmenpräsenz.

4

TEXT FORMATIEREN

Texte spielen auf einer Webseite eine große Rolle. Sie vermitteln dem Besucher nicht nur Informationen, sondern tragen auch wesentlich zum Erscheinungsbild der Internetpräsenz bei.

Die verwendete Schrift ist nur einer der Faktoren, die das Aussehen eines Textes bestimmen. Ebenso wichtig ist die Formatierung des Textes selbst, also die Ausrichtung, Einzüge usw.

Auch für die Textformatierung stellen Ihnen CSS viele Möglichkeiten zur Verfügung.

Text horizontal ausrichten

Standardmäßig werden Texte auf Webseiten links ausgerichtet. Sie können aber auch bestimmen, dass der Text zentriert oder rechts ausgerichtet wird. Blocksatz lässt sich ebenfalls erzwingen. Die Eigenschaft für die Textausrichtung heißt `text-align`.

So wird die horizontale Textausrichtung festgelegt:

1. `text-align:`
   Geben Sie die Eigenschaft `text-align` ein, gefolgt von einem Doppelpunkt.
2. `Ausrichtung;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von dem Semikolon, der die Stilregel abschließt.

Erlaubte Werte sind `left` (links), `center` (zentriert), `right` (rechts) und `justify` (Blocksatz).

Auf den Blocksatz gehen wir auf der nächsten Seite noch näher ein. Schauen wir uns zuerst einmal ein Beispiel für die Ausrichtungen `left`, `center` und `right` an (siehe Listing 4.1). Das Ergebnis im Browser sehen Sie in Abbildung 4.1.

```
<html><head>
<title>Text ausrichten</title>
<style type="text/css"><!--
.eins { font:14pt Arial,sans-serif;
text-align:left; }
.zwei { font:14pt Arial,sans-serif;
text-align:center; }
.drei { font:14pt Arial,sans-serif;
text-align:right; }
--></style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p class="eins">Dieser Absatz ist links
ausgerichtet. Da links dem Standard ent-
spricht, müsste man die Ausrichtung nicht
extra angeben. Rechts entsteht der so
genannte Flattersatz. </p>
<p class="zwei">Dieser Absatz ist zen-
triert ausgerichtet. </p>
<p class="drei">Dieser Absatz ist rechts
ausgerichtet. Bei dieser Ausrichtung ent-
steht der Flattersatz auf der linken Seite
des Abschnitts.</p>
</body></html>
```

Listing 4.1: Text ausrichten

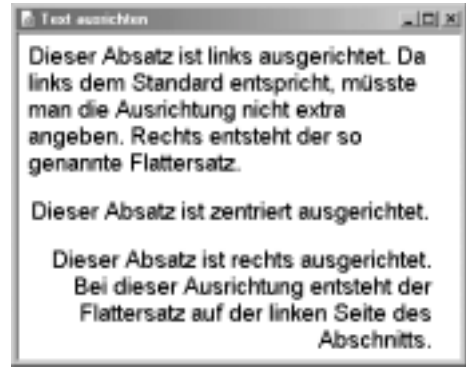

Abbildung 4.1: Die Ausrichtungen `left`, `center` und `right`

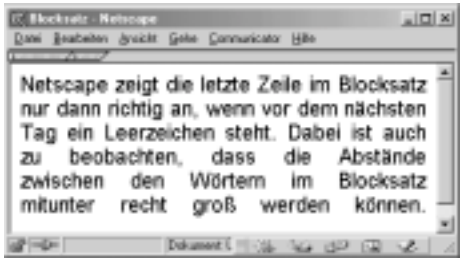

Abbildung 4.2: Hier fehlt das Leerzeichen vor dem nächsten Tag. Netscape zeigt die letzte Zeile daher nicht links ausgerichtet an.

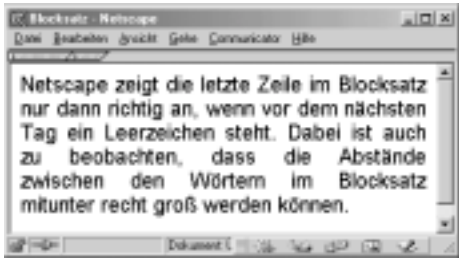

Abbildung 4.3: Die gleiche Seite, wenn das Leerzeichen vor dem nächsten Tag gesetzt wurde. Nun zeigt auch Netscape die letzte Zeile richtig an, also links ausgerichtet.

Blocksatz erzwingen

Bei der Textverarbeitung kennt man den Blocksatz schon lange. Damit wird der Text beidseitig ausgerichtet, was optisch häufig besser wirkt als die Linksausrichtung mit Flattersatz. Die letzte Zeile eines Absatzes wird nur links ausgerichtet. Auch mit CSS haben Sie diese Möglichkeit, indem Sie für `text-align` den Wert `justify` angeben.

Beim Blocksatz ist zu beachten, dass es zu großen Lücken zwischen den Wörtern kommen kann. Es empfiehlt sich daher, den Blocksatz nur in Elementen mit fester Breite anzuwenden, z.B. Tabellenspalten, deren Breite mit Pixelangaben festgelegt ist. Sie sollten auch immer eine Schriftgröße festlegen. So haben Sie die Dehnung der Wortabstände etwas besser unter Kontrolle.

Netscape 4.x nimmt den Blocksatz manchmal zu wörtlich. Die letzte Zeile eines Absatzes und Zeilen mit Umbruch sollten nur links ausgerichtet sein. Netscape zeigt aber auch diese Zeilen im Blocksatz an, wenn vor dem folgenden Tag kein Leerzeichen steht und der Inhalt mehr als ca. 3/4 der Zeilenlänge überschreitet. Achten Sie daher darauf, immer ein Leerzeichen vor dem darauffolgenden Tag zu setzen.

Abbildung 4.2 und Abbildung 4.3 zeigen das Netscape-Verhalten sowie große Wortabstände.

Text vertikal ausrichten

Die vertikale Ausrichtung eines Textes erfolgt standardmäßig immer an der Grundlinie der Schrift. Deutlich wird dieses Verhalten, wenn Sie in einer Zeile verschiedene Schriftgrößen verwenden (siehe Abbildung 4.4).

HTML kennt nur die Hoch- und Tiefstellung von Text, CSS bieten dagegen mehr Möglichkeiten mit der Eigenschaft `vertical-align`. Folgende Werte sind möglich:

- `top` = oben am höchsten Element im Elternelement
- `text-top` = am oberen Rand der Schrift des Elternelements
- `middle` = mittig an der Schrift des Elternelements
- `text-bottom` = am unteren Rand der Schrift des Elternelements
- `bottom` = unten am tiefsten Element im Elternelement
- `baseline` = an der Grundlinie der Schrift (unten, wenn es keine Grundlinie gibt)
- `sub` = tieferstellen, relativ zur Grundlinie
- `super` = höherstellen, relativ zur Grundlinie

Wie sieht in der Praxis der Unterschied zwischen `top` und `text-top` aus? Dazu schauen wir uns Abbildung 4.5 an.

In Abbildung 4.6 sehen Sie die Werte `sub` und `super`.

Abbildung 4.4: Text wird an der Grundlinie ausgerichtet.

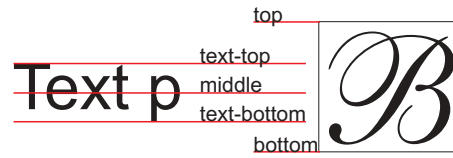

Abbildung 4.5: Der Text ist die Schriftart des Elternelements; das Initial liegt als Grafik vor. Die Werte `top` und `bottom` beziehen sich auf das höchste Element, also die Grafik. `text-top`, `middle` und `text-bottom` werden immer an der Schrift des Elternelements ausgerichtet.

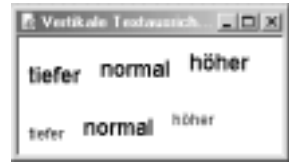

Abbildung 4.6: Oben die CSS-Stile, darunter im Vergleich die Formatierung mit den HTML-Tags `<sub>` und `<sup>`. HTML verkleinert den formatierten Text, CSS nicht.

```
<html><head>
<title>Hochstellung</title>
<style type="text/css">
<!--
p { font:16px Arial,sans-serif; }
.hoch { font:12px Arial,sans-serif;
vertical-align:super; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Die Formel lautet: </p>
<p>a<span class="hoch">2</span> + b
<span class="hoch">2</span> = c
<span class="hoch">2</span> </p>
</body></html>
```

Listing 4.2: Vertikale Textausrichtung

Abbildung 4.7: Darstellung im Browser

So wird die vertikale Textausrichtung festgelegt:

1. `vertical-align:`
   Geben Sie die Eigenschaft `vertical-align` ein, gefolgt von einem Doppelpunkt.
2. `Ausrichtung;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 4.2 zeigt ein Beispiel, das Ergebnis im Opera sehen Sie in Abbildung 4.7.

✓ Tipp:

- `vertical-align` wird von Netsape 4 nicht und vom Internet Explorer nur bedingt unterstützt. In der CSS-Referenz steht detailliert, welche Werte die verschiedenen Browserversionen auf Windows und MAC unterstützen.

Zeichenabstände festlegen

Der Leerraum zwischen den Buchstaben eines Wortes wird im Fachjargon *Kerning* genannt, die Bezeichnung *Zeichenabstand* ist geläufiger. Der Browser legt für jede Schrift den Zeichenabstand automatisch fest. Er hängt von der verwendeten Schriftart und -größe ab. CSS bieten mit der Eigenschaft `letter-spacing` die Möglichkeit, den Zeichenabstand zu definieren, um Effekte oder Hervorhebungen zu erzeugen.

So legen Sie den Zeichenabstand fest:

1. `letter-spacing:`
   Geben Sie die Eigenschaft `letter-spacing` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Als Werte für `letter-spacing` sind nur Maße erlaubt; Prozentangaben dürfen nicht verwendet werden. Sie können auch negative Werte verwenden, um die Wörter zu stauchen.

In Listing 4.3 sehen Sie ein Beispiel, Abbildung 4.8 zeigt das Ergebnis im Opera und IE5.

Netscape 4.x unterstützt `letter-spacing` nicht (siehe Abbildung 4.9). Die Farbformatierung, die zusätzlich vorgenommen wurde, wird aber erkannt. Es funktioniert also nur ein Teil der Stilregel nicht.

```
<html><head>
<title>Zeichenabstände</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif; }
.weit { letter-spacing:10px;
color:#E00000; }
.eng { letter-spacing:-2px;
color:#E00000; }
--></style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Für Hervorhebungen kann man die
Zeichenabstände festlegen.  <br>
<span class="weit">Dehnungen</span> werden
mit positiven Werten erzeugt, negative
Werte verursachen <span class="eng">Stau-
chungen</span>. Netscape 4.x unterstützt
letter-spacing leider nicht. </p>
</body></html>
```

Listing 4.3: Zeichenabstände

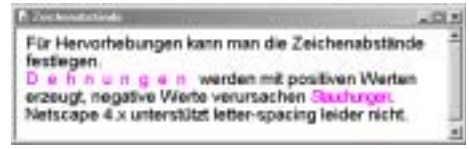

Abbildung 4.8: Opera und IE5 zeigen die Farbe und die definierten Zeichenabstände an.

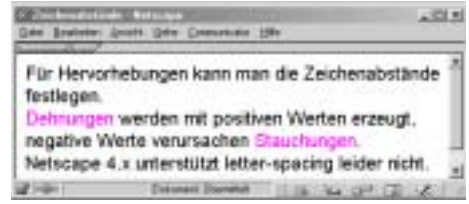

Abbildung 4.9: Netscape zeigt nur die Farbe an, die zusätzlich definiert wurde.

```
<html><head>
<title>Wortabstände</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif; }
.weit { word-spacing:10px; }
.eng { word-spacing:-3px; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p class="weit">Wortabstände im Opera. </
p>
<p class="eng"> Wortabstände im Opera. </
p>
<p> Wortabstände im Opera. </p>
</body></html>
```

Listing 4.4: Wortabstände

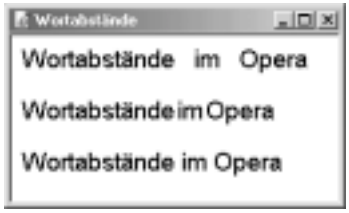

Abbildung 4.10: Im ersten Absatz ist ein positiver Wert definiert, im zweiten ein negativer Wert. Der dritte Absatz ist ohne `word-spacing` gesetzt. IE5 und N4 zeigen `word-spacing` nicht an.

Wortabstände festlegen

Wie bei den Zeichenabständen haben Sie auch für die Wortabstände die Möglichkeit, sie manuell festzulegen. Die Eigenschaft lautet `word-spacing`, wird aber weder vom IE5 noch von Netscape 4.x unterstützt. Opera hingegen zeigt die Eigenschaft an.

So legen Sie die Wortabstände fest:

1. `word-spacing:`
   Geben Sie die Eigenschaft `word-spacing` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Als Werte für `word-spacing` sind nur Maße erlaubt; Prozentangaben dürfen nicht verwendet werden. Sie können auch negative Werte verwenden, um die Abstände zu verringern.

Listing 4.4 zeigt ein Beispiel, das Ergebnis im Opera sehen Sie in Abbildung 4.10.

✓ Tipp:

- Im Blocksatz sollten Sie `word-spacing` nicht verwenden, da dann keine der beiden Eigenschaften richtig angezeigt wird.

Texteinzüge festlegen

Bei mehrzeiligen Texten können Sie für die erste Zeile einen Einzug festlegen. Die Eigenschaft für Einzüge heißt `text-indent`.

So legen Sie einen Einzug fest:

1. `text-indent:`
   Geben Sie die Eigenschaft `text-indent` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Als Werte für `text-indent` sind nur Maße erlaubt. Mit negativen Werten werden Textausrückungen erzeugt. Dabei wird die erste Zeile um das angegebene Maß in den Seitenrand gerückt. Wenn der Rand nicht breit genug ist, schneidet der Browser den Text entsprechend ab.

Listing 4.5 zeigt ein Beispiel. Abbildung 4.11 zeigt eine Seite, bei der der Rand für die Ausrückung zu schmal ist. Die gleiche Seite, diesmal mit ausreichend breitem Rand, sehen Sie in Abbildung 4.12.

✓ Tipp:

- Texteinzüge sind hervorragend geeignet, um lange Texte optisch zu gliedern. Wenn Sie die Einzüge zusammen mit dem Blocksatz verwenden, erzielen Sie sehr professionelle Ergebnisse.

```
<html><head>
<title>Einzüge</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif; }
.einzug { text-indent:30px; }
.negativ { text-indent:-30px; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p class="einzug">Die erste Zeile des
Absatzes wird eingezogen... </p>
<p class="negativ">Hier wird eine
Ausrückung gesetzt... </p>
</body></html>
```

Listing 4.5: Einzüge

Abbildung 4.11: Die Ausrückung wird nicht richtig angezeigt, weil der Seitenrand dafür zu schmal ist.

Abbildung 4.12: Die gleiche Seite, diesmal mit ausreichend breitem Rand

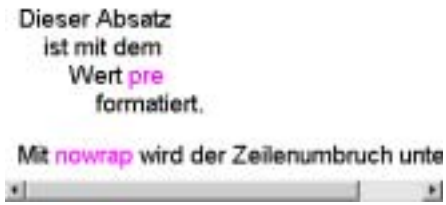

Abbildung 4.13: Opera unterstützt beide Werte.

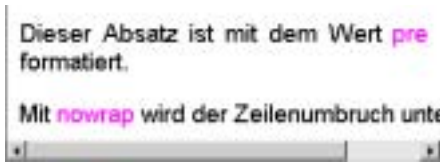

Abbildung 4.14: IE5 unterstützt nur `nowrap`.

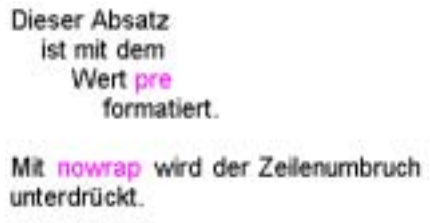

Abbildung 4.15: N4 unterstützt nur `pre`.

Umbrüche und Leerzeichen

Normalerweise werden mehrere Leerzeichen und Zeilenumbrüche, die im Editor eingegeben werden, von den Browsern nicht angezeigt. HTML stellt zwar das Tag `<pre>` zur Verfügung, das die Tastatureingaben so anzeigt, wie sie im Editor vorgenommen werden, die Texte im `<pre>`-Tag werden aber immer in einer Monospace-Schriftart angezeigt.

CSS bieten Ihnen die Möglichkeit, die Eingabe von Leerzeilen und Umbrüchen in beliebigen Tags zu kontrollieren. Die Eigenschaft heißt `white-space`, der Wert `pre`.

`white-space` kennt noch einen anderen Wert: `nowrap`. Damit können Sie den Zeilenumbruch unterdrücken. Die Texte stehen dann in einer Zeile, bei Bedarf erscheinen waagerechte Scrollbalken. Die Vorformatierung der Leerzeichen und Zeilenumbrüche ist dann allerdings nicht mehr möglich.

Opera unterstützt beide Werte (siehe Abbildung 4.13). IE5 definiert nur den Wert `nowrap` (siehe Abbildung 4.14), Netscape 4 hingegen nur `pre` (siehe Abbildung 4.15).

So kontrollieren Sie Umbrüche:

1. `white-space:`
   Geben Sie die Eigenschaft `white-space` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Für `white-space` ist auch der Wert `normal` erlaubt. In Abschnitten, die komplett mit `pre` formatiert sind, können Sie damit für einzelne Passagen festlegen, dass Leerzeichen und Umbrüche ganz normal behandelt werden.

Listing 4.6 zeigt ein Beispiel für den Einsatz von `white-space`. Die Punkte entsprechen den zusätzlich eingegeben Leerzeichen. In Abbildung 4.16 sehen Sie das Ergebnis im Opera.

```
<html><head>
<title>Umbrüche</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif;
text-align:justify; }
.eins { white-space:pre; }

-->
</style></head>
<body>
<p class="eins"><b>
H...E...A...D...L...I...N...E</b>

<span style="white-space:normal;">
<br>

Nun .....folgt ....Text, .in ...
dem ...die ..zusätzlichen ..Leerzeichen
..und ..die...Umbrüche

ignoriert.. werden</span>
</p>
</body></html>
```

Listing 4.6: Umbruchkontrolle mit White-Space

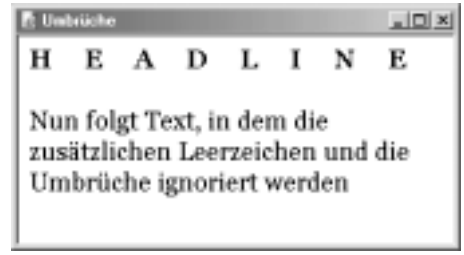

Abbildung 4.16: Die eingegebenen Leerzeichen und Zeilenumbrüche werden im Text unter der Headline ignoriert. Für diesen Bereich wurde mit dem Wert normal die Vorformatierung abgeschaltet.

```
<html><head>
<title>Textdekorationen</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif; }
.unt { text-decoration:underline; }
.ueber { text-decoration:overline; }
.durch { text-decoration:line-through; }
.blinken { text-decoration:blink; }
.keine { text-decoration:none; }
--></style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p> <span class="unt">Unterstrichen</span>
<span class="ueber">überstrichen (nicht
N4)</span> <span class="durch">durchge-
strichen</span> <span class="blinken">
blinkend (nicht IE5)</span><br><br>
<span class="keine">Wenn Sie mit none die
Textdekoration abschalten, werden Links in
etlichen Browsern nicht mehr unterstrichen
angezeigt:
<a href="#">Das ist ein Link </a>
</span>
</body></html>
```

Listing 4.7: Textdekorationen

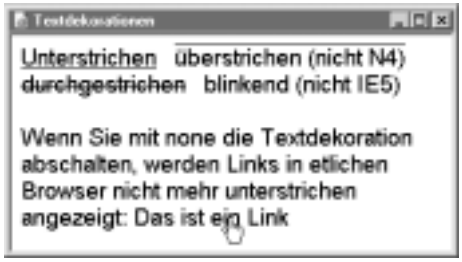

Abbildung 4.17: Dekorativ im Browser

Textdekorationen

Textdekorationen wie Unterstreichungen kann man mit CSS natürlich auch realisieren. Die Eigenschaft heißt `text-decoration`, erlaubte Werte sind `blink` (blinkend), `line-through` (durchgestrichen), `underline` (unterstrichen), `overline` (überstrichen) und `none` (keine).

So legen Sie eine Textdekoration fest:

1. `text-decoration:`
   Geben Sie die Eigenschaft `white-space` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von dem Semikolon, der die Stilregel abschließt.

IE5 stellt blinkenden Text nicht dar, N4 kann den Wert `overline` nicht definieren.

In Listing 4.7 sehen Sie ein Beispiel, Abbildung 4.17 zeigt das Ergebnis im Browser.

✓ Tipps

- Mit dem Wert `none` sollten Sie vorsichtig umgehen. Wenn ein Abschnitt mit diesem Wert formatiert ist, werden in etlichen Browsern auch Links nicht mehr unterstrichen dargestellt. Sie sollten dann in jedem Fall für die Links eigene Formate festlegen (siehe Kapitel 5).
- Unterstrichener Text wird standardmäßig als Link identifiziert. Verwenden Sie den Wert `underline` daher sparsam.
- Blinkender Text wird häufig als störend empfunden, daher sollten Sie auch diese Eigenschaft sparsam einsetzen.

Groß-/Kleinschreibung

Mit der Eigenschaft `text-transform` können Sie die Groß-/Kleinschreibung (Texttransformation) erzwingen. Die Anzeige im Browser erfolgt dann unabhängig davon, wie die Buchstaben tatsächlich im Quelltext stehen.

Erlaubte Werte für `text-transform` sind:

- `uppercase`: nur Großbuchstaben
- `lowercase`: nur Kleinbuchstaben
- `capitalize`: Wortanfänge groß geschrieben
- `none`: wie im Quelltext eingegeben

So legen Sie die Texttransformation fest:

1. `text-transform:`
   Geben Sie `text-transform` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

In Listing 4.8 sehen Sie ein Beispiel. Achten Sie dabei auf die Groß-/Kleinschreibung der Absatzinhalte. Abbildung 4.18 zeigt das Ergebnis im Browser.

✓ Tipp:

- Wenn Sie Überschriften in Großbuchstaben darstellen lassen wollen, sollten Sie die Buchstaben im Quelltext in Großbuchstaben eingeben, damit sie auch von älteren Browsern richtig angezeigt werden.

```
<html><head>
<title>Texttransformation</title>
<style type="text/css">
<!--
p { font:14pt Arial,sans-serif; }
.gross { text-transform:uppercase; }
.klein { text-transform:lowercase; }
.anf { text-transform:capitalize; }
--></style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p class="gross">nur grossbuchstaben </p>
<p class="klein">NuR KleinBUCHstaben </p>
<p class="anf">alle wortanfänge groß </p>
</body></html>
```

Listing 4.8: Texttransformation

Abbildung 4.18: Browserdarstellung

5 Listen, Pseudo-Formate und Maus-Cursor

Listen werden auf Webseiten häufig verwendet. HTML bietet hier einige nützliche Formatierungen an, z.B. die automatische Nummerierung oder Symbole für die Listenpunkte. CSS bieten Ihnen mehr Möglichkeiten, Listen nach Ihren Wünschen zu formatieren, z.B. die Liste aus- statt einzurücken oder eigene Listensymbole zu definieren.

Netscape 4 ist bei der Listenformatierung wieder der Wermutstropfen; einige CSS-Formate werden von diesem Browser noch nicht unterstützt.

Gleiches gilt bei Pseudo-Formaten für Links und Absätze sowie für das Aussehen des Mauscursors, das Sie in CSS gezielt festlegen können.

Dies sollte Sie aber nicht davon abhalten, diese Formate anzuwenden, da IE und Opera die meisten Eigenschaften ohne Probleme anzeigen. Netscape 4 zeigt die Elemente auch weiterhin an, nur eben ohne die Formatierungen.

Allgemeines zu Listen

Die CSS-Angaben für Listen beziehen sich speziell auf ungeordnete und geordnete Listen. Beide Listentypen werden nach demselben Schema aufgebaut: Der Tag-Container für den gewünschten Listentyp umschließt die Tags für die Listenpunkte (siehe Abbildung 5.1).

Ungeordnete Listen werden mit Bullets (Symbolen) als Aufzählungszeichen angezeigt, geordnete Listen werden automatisch durchnummeriert. Die Art der Aufzählungszeichen kann festgelegt werden (siehe Abbildung 5.2).

Ungeordnete Liste:

```
<ul>
<li>Listenpunkt </li>
<li>Listenpunkt </li>
</ul>
```

Geordnete Liste:

```
<ol>
<li>Listenpunkt </li>
<li>Listenpunkt </li>
</ol>
```

Abbildung 5.1: Aufbau von Listen

✓ Tipps:

- Die Anzeige der Symbole für ungeordnete Listen ist je nach Betriebssystem und Browser verschieden. Opera zeigt den Typ `square` z.B. als ungefülltes Rechteck, während der IE ein gefülltes Rechteck anzeigt.
- Neuere Browser zeigen in beiden Listentypen auch die Bullets des jeweils anderen Typs an. Wenn Sie also in einer ungeordneten Liste als Bullet-Typ `lower-alpha` für Kleinbuchstaben angeben, werden diese angezeigt, obwohl sie eigentlich nicht für die ungeordnete Liste vorgesehen sind. Sie sollten sich aber an die in Abbildung 5.2 gezeigten Konventionen halten, damit auch Besucher mit älteren Browsern die richtigen Bullet-Zeichen sehen.

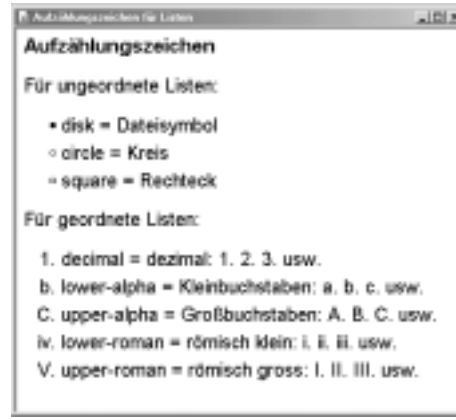

Abbildung 5.2: Aufzählungszeichen für Listen

```
<html><head>
<title>Bulletstile</title>
<style type="text/css">
<!--
ul { list-style-type:circle; }
ol { list-style-type:decimal; }
.extra1 { list-style-type:square; }
.extra2 { list-style-type:upper-alpha; }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
Ungeordnete Liste:<br>
<ul><li>Thema 1</li>
<li>Thema 2</li>
<li class="extra1">Extra Thema</li>
</ul>
Geordnete Liste:<br>
<ol><li>Thema 1</li>
<li>Thema 2</li>
<li class="extra2">Extra Thema</li>
</ol></body></html>
```

Listing 5.1: Bullet-Stile

Abbildung 5.3: Bullets im Browser

Listen: Bullet-Typ festlegen

Die Eigenschaft für die Aufzählungszeichen (Bullets) heißt `list-style-type`. Wie schon erwähnt, richtet sich der Wert nach dem Listentyp.

Für eine ungeordnete Liste `<ul>` sind die Werte `disk`, `circle` und `square` erlaubt, für geordnete Listen `<ol>` die Werte `decimal`, `lower-alpha`, `upper-alpha`, `lower-roman` und `upper-roman`. Es gibt für geordnete Listen noch mehr Werte, z.B. `hebrew` für hebräische Zeichen, diese werden jedoch von den Browsern nicht unterstützt und finden hier daher keine weitere Erwähnung. Für beide Listentypen kann mit `none` die Anzeige der Aufzählungszeichen unterdrückt werden.

So legen Sie den Bullet-Typ fest:

1. `list-style-type:`
   Geben Sie die Eigenschaft `list-style-type` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

In Listing 5.1 sehen Sie den Quellcode für zwei Listen mit jeweils verschiedenen Bullet-Stilen; Abbildung 5.3 zeigt das Ergebnis im Browser.

Listen: Bullet-Position

Wenn ein Listenthema sich über mehrere Zeilen erstreckt, wird das Bullet außerhalb der Liste angezeigt (siehe Abbildung 5.4).

Sie können die Position des Bullets mit der Eigenschaft `list-style-position` beeinflussen. Erlaubte Werte sind `outside`, was dem Standard entspricht und `inside`. Mit `inside` beginnt die nächste Zeile unter dem Bullet, nicht unter dem Text der ersten Zeile. Der Einzug der Liste wird dabei nicht verändert, sie wird genauso weit eingerückt wie bei der Standardposition. Das Bullet selbst wird in die Liste gerückt. Das Mischen beider Positionen ist ebenfalls möglich (siehe Abbildung 5.5).

Netscape4 unterstützt die Positionen nicht, das Bullet wird immer außerhalb der Liste angezeigt.

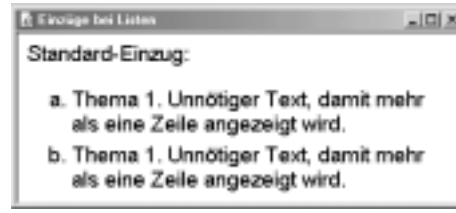

Abbildung 5.4: Standardposition des Bullets

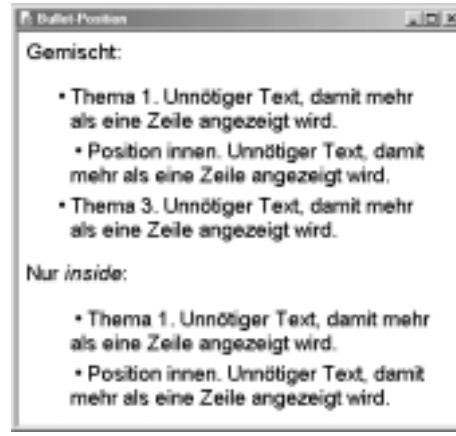

Abbildung 5.5: Eingerückte Bullets

```
<html><head>
<title>Bullet-Position</title>
<style type="text/css">
<!--
body { font:14pt Arial,sans-serif; }
ul { list-style-position:outside;
list-style-type:disk; }
.in { list-style-position:inside; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
Gemischt: <br>
<ul>
<li>Thema 1. </li>
<li class="in">Position innen. </li>
<li>Thema 3. </li>
</ul>
Nur <i>inside</i>:<br>
<ul class="in">
<li>Thema 1. </li>
<li>Thema 2. </li>
</ul></body></html>
```

Listing 5.2: XXX

So legen Sie die Bullet-Position fest:

1. `list-style-position:`
   Geben Sie die Eigenschaft `list-style-position` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 5.2 zeigt ein Beispiel, das Ergebnis (mit etwas mehr Text) sehen Sie in Abbildung 5.5 auf der vorherigen Seite.

Listen: Grafiken für Bullets

Für ungeordnete Listen können Sie eigene Grafiken als Aufzählungszeichen festlegen. Die Eigenschaft heißt `list-style-image`, als Wert wird immer `url(grafikdatei)` angegeben. Netscape 4 unterstützt diese Eigenschaft nicht.

So legen Sie eine Bullet-Grafik fest:

1. `list-style-image:`
   Geben Sie die Eigenschaft `list-style-image` ein, gefolgt von einem Doppelpunkt.
2. `url(`
   Schreiben Sie dahinter den Wert `url`, dahinter folgt, ohne Leerzeichen, die öffnende runde Klammer.
3. `verzeichnis/grafik.gif);`
   Geben Sie den Pfad der gewünschten Grafikdatei an, gefolgt von der schließenden runden Klammer und dem Semikolon, der die Stilregel abschließt.

Ein Beispiel sehen Sie in Listing 5.3. Abbildung 5.6 zeigt das Ergebnis im Browser.

✓ Tipp:

- Wenn Sie eigene Bullet-Grafiken einsetzen, sollten Sie immer auch den Bullet-Typ angeben. Er wird in Browsern, die Bullet-Grafiken unterstützen, in diesem Fall ignoriert, stellt aber sicher, dass in anderen Browsern einheitliche Bullets angezeigt werden.

```
<html><head>
<title>Bullet-Grafiken</title>
<style type="text/css">
<!--
body { font:13pt Arial,sans-serif; }
ul { list-style-type:square;
list-style-image:url(pfeil.gif); }
--></style>
</head><body bgcolor="#FFFFFF"
text="#000000">
Unsere Angebote: <br>
<ul>
<li>Desktop-Computer</li>
<li>Notebooks</li>
<li>Serversysteme</li>
</ul></body></html>
```

Listing 5.3: Bullet-Grafiken

Abbildung 5.6: Bullet-Grafiken im Browser

```
<html><head>
<title>Kurzschrift</title>
<style type="text/css">
<!--
body { font:13pt Arial,sans-serif; }
ul { list-style:disk url(pfeil.gif);
font:italic 13pt Arial,sans-serif; }
.extra { list-style:square url(pfeil-
rot.gif);
font:bold 13pt Arial,sans-serif; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
Unsere Angebote: <br>
<ul>
<li> Desktop-Computer</li>
<li> Notebooks</li>
<li class="extra"> Serversysteme</li>
</ul>
</body></html>
```

Listing 5.4: Kurzschrift für Listen

Abbildung 5.7: Browseranzeige

Kurzschrift für Listen

Wie bei den Schrifteigenschaften gibt es auch für Listen eine Kurzschrift, mit der die drei Eigenschaften `list-style-type`, `list-style-position` und `list-style-image` zusammengefasst werden können. Der Befehl lautet `list-style`.

So wenden Sie die Kurzschrift an:

1. `list-style:`
   Geben Sie die Eigenschaft `list-style` ein, gefolgt von einem Doppelpunkt.
2. `Typ Position url(grafikdatei);`
   Schreiben Sie dahinter die gewünschten Werte, getrennt durch Leerzeichen. Abgeschlossen wird die Stilregel wie immer durch ein Semikolon.

Listing 5.4 zeigt die Kurzschrift im Einsatz. In Abbildung 5.7 sehen Sie das Ergebnis im Browser.

✓ Tipp:

- Bei der Kurzschrift für Schrifteigenschaften spielt die Reihenfolge der Eigenschaften eine Rolle, bei der Kurzschrift für Listen ist die Reihenfolge egal. Eigenschaften, die nicht benötigt werden, können weggelassen werden.

Maus-Cursor nach Wahl

Sie können für bestimmte Bereiche der Webseite das Aussehen des Maus-Cursors beeinflussen. Ein Link zum Seitenanfang kann z.B. mit dem nach oben zeigenden Maussymbol versehen werden.

Die Eigenschaft zum Festlegen des Maus-Cursors heißt `cursor`. Die erlaubten Werte sehen Sie in Abbildung 5.8. Beim Einsatz sollte man bedenken, dass viele Mauscursor für den Besucher mit einer bestimmten Bedeutung verknüpft sind. Ein Cursor, der das Verschieben eines Bereichs andeutet, sollte auch nur angezeigt werden, wenn der Bereich tatsächlich verschoben werden kann, z.B. bei DHTML-Ebenen, die der Besucher bewegen kann.

Je nach Browser und Betriebssystem werden die Cursor-Symbole unterschiedlich angezeigt. Das ist recht sinnvoll, denn ein Mac-Nutzer könnte z.B. mit den bei Windows üblichen Symbolen wenig anfangen.

✓ Tipp:

- Leider bieten Netscape 4 und Opera für die Mauscursor keine Unterstützung an. IE5 und Netscape 6 hingegen zeigen die Cursor an. Da laut Statistik zum Zeitpunkt der Drucklegung dieses Buchs immerhin mehr als die Hälfte aller Surfer den IE5 benutzt, sollten Sie diesen Besuchern die visuelle Unterstützung durch verschiedene Mauscursor nicht vorenthalten. In den anderen Browsern werden die Standard-Cursor angezeigt.

	auto		w-resize
	default		nw-resize
	pointer		n-resize
	hand		ne-resize
	crosshair		e-resize
	text		se-resize
	help		s-resize
	wait		sw-resize

Abbildung 5.8: So sehen die Mauscursor in Windows mit IE5 aus. Der Wert `pointer` wird vom IE nicht unterstützt, in diesem Browser heißt der Wert `hand`. Das Aussehen der beiden Werte ist gleich. Beim Wert `auto` legt der Browser je nach Bereich das Aussehen des Cursors automatisch fest. Dieser Wert entspricht dem Standard ohne CSS-Formatierung.

```
<html><head>
<title>Mauscursor</title>
<style type="text/css">
<!--
body { font:13pt Arial,sans-serif; }
.hilfe { cursor:help; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Wenn Sie Hilfe zur Bedienung des Forums
benötigen, rufen Sie bitte die
<a class="hilfe" href="hilfe.html">
Hilfedatei</a> auf.
</p></body></html>
```

Listing 5.5: Maus-Cursor

Abbildung 5.9: Für den Link zur Hilfedatei ist der Cursor-Wert `help` festgelegt.

So legen Sie einen Maus-Cursor fest:

1. `cursor:`
   Geben Sie die Eigenschaft `cursor` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 5.5 zeigt ein Beispiel, in Abbildung 5.9 sehen Sie das Ergebnis im Browser.

✓ Tipp:

- Theoretisch ist es auch möglich, mit dem Befehl `cursor:url(grafikdatei);` eine eigene Grafik als Mauscursor festzulegen. Dieser Wert wird aber bisher von keinem Browser unterstützt.

Pseudo-Elemente formatieren

Pseudo-Elemente sind Teile einer Webseite, die sich nicht durch ein eindeutiges HTML-Tag ausdrücken lassen, z.B. "der erste Buchstabe", "die erste Zeile" oder "ein besuchter Link". Die allgemeine Syntax für Pseudo-Formate wurde bereits in Kapitel 2 besprochen.

Etliche Pseudo-Elemente, für die CSS bereits Befehle anbietet, werden von den gängigen Browsern nicht unterstützt. Dazu gehören z.B. Formate für Drucklayouts. Auf diese Formate wird in diesem Buch nicht näher eingegangen.

Welche Pseudo-Elemente Sie auf den nächsten Seiten näher kennen lernen werden, sehen Sie in Tabelle 5.1.

ELEMENT	BEDEUTUNG
`a:link`	Hyperlink zu noch nicht besuchten Seiten
`a:visited`	Hyperlink zu bereits besuchten Seiten
`a:hover`	Hyperlink, der mit der Maus überfahren wird
`a:active`	Hyperlink, der gerade angeklickt wird
`:first-letter`	erstes Absatzzeichen
`:first-line`	erste Absatzzeile

Tabelle 5.1: Manche Pseudo-Elemente sind einem bestimmten Tag zugeordnet, z.B. `:link`. Andere Elemente können in diversen Tags verwendet werden. Sie stehen in der Tabelle ohne HTML-Selektor vor dem Schlüsselwort.

✓ Tipp:

- Bei den Angaben `a:link`, `a:visited`, `a:hover` und `a:active` ist das eigentliche Element nur das, was hinter dem HTML-Selektor `a` steht, also z.B. `:link`. Da die Pseudo-Elemente für Hyperlinks aber fest an den HTML-Selektor `a` gebunden sind, sehen wir `a` als Bestandteil des Pseudo-Elements an.

Wichtige Reihenfolge

Damit die Pseudo-Formate für Links richtig angezeigt werden, muss in der CSS-Definition die folgende Reihenfolge eingehalten werden:

```
a:link
a:visited
a:hover
a:active
```

Wenn Sie eines der Formate nicht benötigen, können Sie es weglassen.

Abbildung 5.10: IE zeigt `a:active` noch an, wenn Sie auf die Seite zurückkehren und noch keinen anderen Link angeklickt haben. Andere Browser zeigen das Format nur dann an, wenn die Maustaste über dem Link gedrückt wird.

Links formatieren

Standardmäßig werden Hyperlinks unterstrichen dargestellt. Für die Farben verwenden die Browser die Standardeinstellungen, solange keine anderen Eigenschaften festgelegt werden. Mit den Pseudo-Formaten für Links können Sie nicht nur die Farben für verschiedene Linkzustände festlegen, sondern auch spezielle Schriftformate festlegen, die Unterstreichung unterdrücken oder Hintergründe definieren, die einen Textmarker-Effekt erzeugen.

Bei den Links gibt es verschiedene Pseudo-Elemente. Man könnte sie auch als Zustände bezeichnen:

- `a:link` steht für Links zu Seiten, die noch nicht besucht wurden.
- `a:visited` zeigt dem Besucher, welche Links bereits aufgerufen wurden.
- `a:hover` für Links, über die man mit der Maus streicht, ohne sie anzuklicken.
- `a:active` steht für den Link, den der Besucher gerade anklickt.

✓ Tipps:

- Netscape 4 interpretiert die Pseudo-Elemente `a:hover` und `a:active` nicht.
- `a:active` wird von den Browsern unterschiedlich definiert. Opera zeigt das Format nur an, während die Maustaste über dem Link gedrückt wird. IE zeigt das Format auch noch an, wenn Sie mit dem *Zurück*-Button auf die Seite zurückkehren und noch keinen anderen Link angeklickt haben (siehe Abbildung 5.10).

So legen Sie die Link-Formate fest:

1. `a:link { Definitionen; }`
   Geben Sie das Element `a:link` an. Dahinter stehen in geschweiften Klammern die gewünschten Stildefinitionen.
2. `a:visited { Definitionen; }`
   In der nächsten Zeile geben Sie das Element `a:visited` an. Auch hier geben Sie in geschweiften Klammern die gewünschten Stildefinitionen an.
3. `a:hover { Definitionen; }`
   Als Nächstes schreiben Sie das Element `a:hover`, wieder gefolgt von den geschweiften Klammern, in denen die Definitionen stehen.
4. `a:active { Definitionen; }`
   Die letzte Angabe ist das Element `a:active` mit den entsprechenden Definitionen.

Listing 5.6 zeigt ein Beispiel. In Abbildung 5.11 sehen Sie das Ergebnis im Browser, nachdem Link1 und Link3 besucht wurden.

✓ Tipp:

- Es können auch Klassen definiert werden, um die Links in verschiedenen Bereichen der Webseite mit unterschiedlichen Formaten zu versehen. Mehr dazu erfahren Sie im Abschnitt *Klassen für Pseudo-Formate* später in diesem Kapitel.

```
<html><head>
<title>Linkformate</title>
<style type="text/css"><!--
p { font:12pt Arial,sans-serif; }
a:link { color:#E00000;
text-decoration:underline; }
a:visited { color:#606060;
text-decoration:none; }
-->
</style></head>
<body bgcolor="#FFFFFF" text="#000000">
<p>
<a href="#L1">Link 1</a><br>
<a href="#L2">Link 2</a><br>
<a href="#L3">Link 3</a><br>
<a href="#L4">Link 4</a><br>
</p></body></html>
```

Listing 5.6: Linkformate

Abbildung 5.11: Browseranzeige

Abbildung 5.12: Der Browser legt anhand des automatischen Textumbruchs am rechten Fensterrand fest, welche Inhalte mit der Formatierung für `:first-line` angezeigt werden.

Abbildung 5.13: Die gleiche Datei, nachdem der Anzeigebereich verbreitert wurde.

Abschnittsanfänge hervorheben

Bei längeren Fließtexten können Sie die erste Zeile oder das erste Zeichen eines Abschnitts hervorheben. Die Pseudo-Formate hierfür lauten `:first-line` (die erste Zeile) und `:first-letter` (das erste Zeichen). Netscape 4 unterstützt leider keines der beiden Pseudo-Formate.

✓ Tipps:

- Im Gegensatz zu den Link-Formaten sind `:first-line` und `:first-letter` nicht an ein bestimmtes Tag gebunden.
- Beide Pseudo-Formate können nur in Block-level-Tags verwendet werden, nicht in Inline-Tags oder Replaced-Tags.
- Sie können die beiden Formate kombinieren, die Reihenfolge in der CSS-Definition ist dabei egal.
- Bei `:first-line` bestimmt immer die Breite des Anzeigebereichs, welche Inhalte noch mit der Formatierung angezeigt werden (siehe Abbildung 5.12 und Abbildung 5.13).
- Das Tag `<br>` beendet in jedem Fall die erste Zeile und damit das Format `:first-line`.
- Es können auch Klassen definiert werden, um die Abschnittsanfänge für ein Tag in verschiedenen Bereichen der Webseite mit unterschiedlichen Formaten zu versehen. Mehr dazu erfahren Sie im Abschnitt *Klassen für Pseudo-Formate* später in diesem Kapitel.

So legen Sie das Aussehen der ersten Zeile fest:

1. `p`
   Geben Sie den HTML-Selektor an, auf den das Pseudo-Format angewendet werden soll. Für das Beispiel wurde `p` gewählt.
2. `:first-line`
   Schreiben Sie direkt dahinter das Element `:first-line`, gefolgt von einem Leerzeichen. Achten Sie darauf, dass das Element von einem Doppelpunkt eingeleitet wird.
3. `{ Definitionen; }`
   In den geschweiften Klammern folgen nun wie üblich die Stildefinitionen.

Listing 5.7 zeigt ein Beispiel, in Abbildung 5.14 sehen Sie das Ergebnis im Opera.

Netscape 4 unterstützt `:first-line` nicht.

```
<html><head>
<title>Erste Zeile</title>
<style type="text/css">
<!--
p { font:12pt Arial,sans-serif; }
p:first-line { font:14pt Arial,sans-serif;
color:#0000C0; }
-->
</style></head><body bgcolor="#FFFFFF"
text="#000000">
<p>In diesem Absatz ist die erste Zeile
hervorgehoben. </p>
</body></html>
```

Listing 5.7: Erste Zeile hervorheben

Abbildung 5.14: Browseranzeige

```
<html><head>
<title>Erstes Zeichen</title>
<style type="text/css">
<!--
p { font:12pt Arial,sans-serif; }
p:first-letter { font:italic 18pt
Arial,sans-serif; color:#ff0000; }
-->
</style></head><body bgcolor="#FFFFFF"
text="#000000">
<p>Hier ist das erste Zeichen hervorgeho-
ben. </p>
<p>In diesem Absatz auch. </p>
</body></html>
```

Listing 5.8: Erstes Zeichen hervorheben

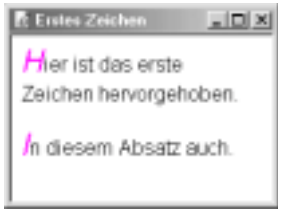

Abbildung 5.15: Browseranzeige

So legen Sie das Aussehen des ersten Zeichens fest:

1. `p`
   Geben Sie den HTML-Selektor an, auf den das Pseudo-Format angewendet werden soll. Für das Beispiel wurde `p` gewählt.
2. `:first-letter`
   Schreiben Sie direkt dahinter das Element `:first-letter`, gefolgt von einem Leerzeichen. Achten Sie darauf, dass das Element von einem Doppelpunkt eingeleitet wird.
3. `{ Definitionen; }`
   In den geschweiften Klammern folgen nun wie üblich die Stildefinitionen.

Listing 5.8 zeigt ein Beispiel, in Abbildung 5.15 sehen Sie das Ergebnis im Browser.

Netscape 4 unterstützt `:first-letter` nicht.

Klassen für Pseudo-Elemente

Häufig möchte man ein Pseudo-Element einsetzen, aber nicht alle betroffenen Tags damit formatieren, oder es sollen für Pseudo-Elemente unterschiedliche Formate definiert werden. Für solche Zwecke können Sie mit Klassen für die Pseudo-Elemente verschiedene Formate festlegen. Der Klassenname wird dabei zwischen dem HTML-Selektor und dem Pseudo-Element eingefügt.

So legen Sie eine Klasse für ein Pseudo-Element fest:

1. `HTML-Selektor`
   Geben Sie den HTML-Selektor an, auf den die Klasse angewendet werden soll. Für das Beispiel wurde `p` gewählt.
2. `.Klassenname`
   Direkt dahinter schreiben Sie den gewünschten Klassennamen, er wird durch einen Punkt eingeleitet.
3. `:Pseudo-Element`
   Schreiben Sie hinter den Klassennamen ohne Leerzeichen das Pseudo-Element. Denken Sie daran, dass die Elemente mit einem Doppelpunkt eingeleitet werden.
4. `{ Definitionen; }`
   In den geschweiften Klammern folgen nun wie üblich die Stildefinitionen.
5. Wiederholen Sie die Schritte 1 bis 4 für alle gewünschten Pseudo-Klassen.

Listing 5.9 zeigt verschiedene Definitionen für Pseudo-Klassen, die in einer externen CSS-Datei notiert wurden.

```
/* Besondere Links in roter Schrift: */

a.rot:link { color:#ff0000; }
a.rot:visited { color:#ff0000;
text-decoration:none; }

/* Fließtext-Standard für Links */

a:link { color:#000000; }
a:visited { color:#606060; }

//Absatzstandard
p { font:12pt Arial,sans-serif; }

/*Absätze mit blauem Initial: */

p.blau:first-letter {
font:italic 20pt Arial,sans-serif;
color:#0000ff; }

/*Absätze mit rotem Initial: */

p.rot:first-letter {
font:italic 20pt Arial,sans-serif;
color:#ff0000; }
```

Listing 5.9: Pseudo-Klassen in einer externen CSS-Datei. Sie können die Pseudo-Klassen natürlich auch mit der internen Methode direkt im Head der Webseite definieren.

✓ Tipp:

- Auch bei Pseudo-Klassen können Klassennamen mehrfach verwendet werden, wenn sie unterschiedlichen Tags bzw. Pseudo-Elementen zugeordnet sind.

```
<html><head>
<title>Pseudo-Klassen</title>
<link rel="stylesheet" href="pseudo.css"
type="text/css">
</head><body bgcolor="#FFFFFF"
text="#000000">
<p class="blau">Dieser Absatz beginnt mit
einem blauen Initial. Hier finden wir auch
einen <a href="seite1.html" class="rot">
Link</a>, der rot dargestellt wird.</p>
<p class="rot">In diesem Absatz ist das
erste Zeichen rot hervorgehoben. Der fol-
gende <a href="#">Link</a> hat keine
Klasse und wird daher im definierten Stan-
dard angezeigt. </p>
</body></html>
```

Listing 5.10: Zweimal die Klasse rot, beide haben aber völlig unterschiedliche Bedeutung, da sie in der CSS-Datei (siehe Listing 5.9) anderen Pseudo-Klassen zugeordnet sind.

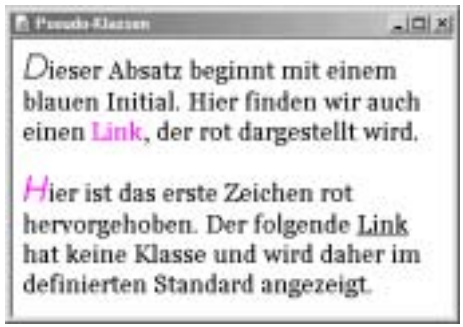

Abbildung 5.16: Browseranzeige

So wenden Sie eine Pseudo-Klasse an:

1. `<p class="blau">`
   In dem Tag, auf das die Pseudo-Klasse angewendet werden soll, notieren Sie das Attribut `class`. Als Wert wird zwischen den Anführungszeichen der Name der Klasse angegeben.
2. `Text </p>`
   Schreiben Sie dann den Text, auf den das Format angewendet werden soll, und setzen Sie danach das End-Tag.

In Listing 5.10 sehen Sie den Quelltext der HTML-Datei, Abbildung 5.16 zeigt das Ergebnis im Browser. Die Definitionen sind in einer externen CSS-Datei notiert (siehe Listing 5.9).

✓ Tipps:

- Wenn Sie ein Tag mit einer Pseudo-Klasse formatieren, wird beim `class`-Attribut als Wert nur der Klassenname angegeben. Weder der HTML-Selektor noch das Pseudo-Element tauchen hier auf. Ebensowenig der Punkt, der in der Stilregel den Klassennamen einleitet.
- Mit den Pseudo-Klassen können Sie Links in einem Navigationsmenü anders darstellen als die Links im Fließtext.
- Wenn Sie für Absätze Initialen definieren, eignen sich die Pseudo-Klassen hervorragend, um z.B. Initialen in verschiedenen Farben darzustellen.
- Sie können Pseudo-Klassen natürlich intern im Head-Bereich der Seite definieren. Sinnvoller ist aber der Einsatz mit der externen Methode, da für Pseudo-Klassen schon etwas mehr Quellcode erforderlich ist. Mit der externen Methode halten Sie den Quelltext der HTML-Seite übersichtlicher und auch kürzer, was Ladezeiten spart.

6

ELEMENTHINTERGRÜNDE

Die einzigen Elemente, denen Sie mit HTML eine Hintergrundfarbe oder -grafik zuteilen können, sind der Seiten-Body und Tabellen.

Mit CSS können Sie jedem Element einen Hintergrund geben. Für Hintergrundgrafiken haben Sie zusätzlich die Möglichkeit, die Kachelung zu unterdrücken, die oft störend wirkt.

Besonders geeignet sind Hintergründe natürlich für Blocklevel-Tags, da diese einen Abschnitt bilden. Aber auch in Inline-Tags kann eine Hintergrundfarbe ganz nützlich sein, wenn es um Hervorhebungen geht. So lassen sich in Anleitungen und Dokumentationen z.B. Textmarker-Effekte realisieren.

Was sind Elemente?

Um zu verstehen, wie die Stylesheets mit Elementen umgehen, klären wir zunächst einmal, was Elemente eigentlich sind. Dieses Wissen benötigen Sie auch in den nachfolgenden Kapiteln.

Ein Element ist ein Stück HTML-Code, das eine Einheit bildet. Das kann z.B. ein Absatz oder eine Grafik sein:

```
<p>Ein Absatz</p>

<img src="grafik.gif">
```

Diese beiden Beispiele sind Elemente, die keine anderen Elemente enthalten. Bei der Grafik wäre das gar nicht möglich. `<img>` gehört zu den Replaced-Tags, die keine anderen Tags enthalten können (siehe Kapitel 1, *Welche Stile für welche Tags?*).

Häufig werden Elemente aber auch in andere Elemente eingebettet. Man nennt das auch verschachteln:

```
<p>Ein Absatz mit <b>anderen</b> Elementen,
z.B. <i>kursivem Text</i> und einer Grafik:
<br>
<img src="grafik.gif"></p>
```

Der Absatz bildet das *Elternelement* für die eingebetteten Elemente, in diesem Fall sind das fetter Text, kursiver Text und eine Grafik. Zusammen mit dem umgebenden Text bilden Sie den Inhalt des Absatzes.

Tags und Container

Dieser Infokasten ist Ihnen bereits aus Kapitel 1 bekannt, hier noch einmal zur Auffrischung:

Häufig ist von Tags die Rede, obwohl eigentlich ein HTML-Container gemeint ist.

Worin liegt der Unterschied und was ist eigentlich ein Container? Ein Tag ist ein HTML-Befehl der in den spitzen Klammern steht, z.B. `<img>` oder `<b>`. Zu vielen Tags gehört ein End-Tag, das gebildet wird, indem man den Slash / voranstellt: `</b>`. Start- und End-Tag bilden zusammen den Container: `<b>Containerinhalt</b>`.

Diese Container werden in der Praxis schlicht als Tags bezeichnet. Auch in diesem Buch wird dieser Begriff verwendet. Wenn vom `<p>`-Tag die Rede ist, kann also der Container `<p>…</p>` gemeint sein.

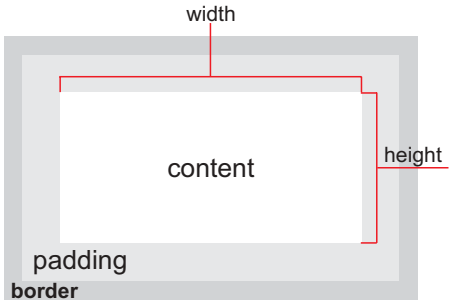

Abbildung 6.1: Das Element selbst besteht aus `border`, `padding` und `content`.

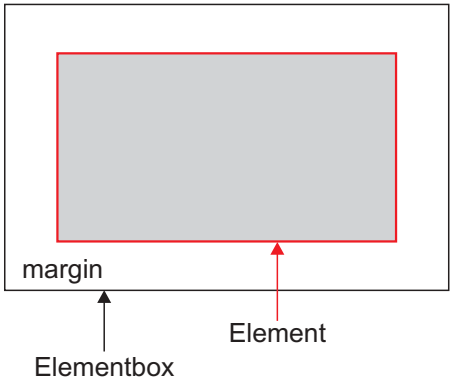

Abbildung 6.2: Zur Elementbox gehört auch der Rand (`margin`).

Die Elementbox

Man kann ein Element mit einem Rechteck vergleichen, das einen bestimmten Raum auf der Webseite einnimmt. Im Fachjargon spricht man aber nicht von einem Rechteck, sondern von der Elementbox oder kurz Box.

Egal, um welches Element es sich handelt, es gibt Bestandteile, die jede Box besitzt, nachdem das Element auf der Webseite platziert wurde. CSS verwendet die englischen Begriffe für die Box-Bestandteile:

- *content* steht für die Inhalte des Elements, also z.B. Text und Grafiken. *width* (Breite) und *height* (Höhe) sind die Abmessungen der Inhalte.
- *padding* ist der Abstand zwischen Inhalt und Rahmen.
- *border* ist der Rahmen, eine Linie, die alle Seiten eines Elements umschließt. Der Rahmen ist die äußere Begrenzung des eigentlichen Elements. Standardmäßig ist er unsichtbar.
- *margin* ist der Rand, der zwischen dem Rahmen und anderen Elementen liegt. Er gehört zur Elementbox, ist aber selbst kein Bestandteil des Elements.

In Abbildung 6.1 sehen Sie die Bestandteile des eigentlichen Elements. Abbildung 6.2 zeigt die gesamte Elementbox, zu der außer `content`, `padding` und `border` auch `margin` gehört.

Allgemeines zu Hintergründen

Hintergründe wirken sich grundsätzlich auf das Element aus, nicht auf die Box. Ausnahme ist hier der Body der Seite. Beim `<body>`-Tag erstreckt sich ein Hintergrund auch auf den `margin`-Bereich, also auf die ganze Elementbox.

Der Standard jeden Elements in Bezug auf den Hintergrund ist *none*, also keiner. Man könnte auch sagen, dass jedes Element durchsichtig ist, solange kein Hintergrund angegeben wird.

Daraus resultiert, dass der Hintergrund eines Elements "durchscheint", wenn dieses Element von anderen Elementen überlagert wird, die keinen eigenen Hintergrund haben.

Am deutlichsten wird dieses Verhalten, wenn das Elternelement, z.B. der Seiten-Body, einen Hintergrund besitzt. Er scheint durch alle Elemente hindurch, die keinen eigenen Hintergrund besitzen (siehe Abbildung 6.3).

Aber auch, wenn sich durch Positionierungen (siehe Kapitel 8) Elemente überlappen, die nicht ineinander verschachtelt sind, ist der Hintergrund des unten liegenden Elements zu sehen, wenn das darüber liegende Element keinen Hintergrund hat.

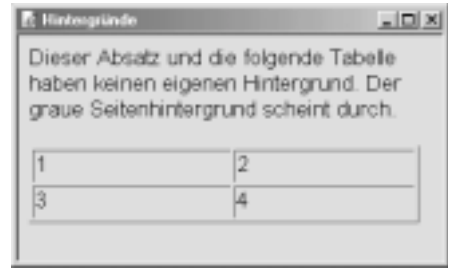

Abbildung 6.3: Absatz und Tabelle haben keinen eigenen Hintergrund; der Seitenhintergrund ist zu sehen.

Abbildung 6.4: Netscape 4 zeigt in Blocklevel-Tags die Hintergrundfarbe nicht richtig an.

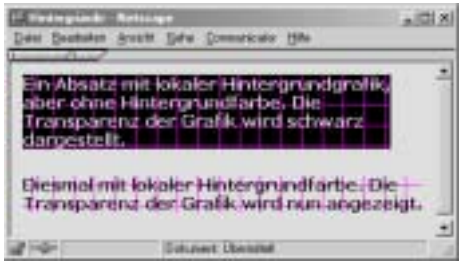

Abbildung 6.5: Auch bei Hintergrundgrafiken mit transparenten Bereichen gibt es in Netscape 4 Probleme.

Netscape 4 und Hintergründe

Bei Hintergründen bringt Netscape 4 mehrere Bugs mit, die in der neuen Version 6 zum Glück nicht mehr vorhanden sind.

Hintergrundfarben sollen nach CSS-Standard in Blocklevel-Tags wie `<p>` über die zur Verfügung stehende Breite als Block angezeigt werden. Netscape zeigt die Hintergrundfarbe aber nur hinter der Schrift an (siehe Abbildung 6.4). Bei mehrzeiligen Abschnitten kann man sich mit dem Befehl `border:none` behelfen, bei einzeiligen Absätzen hilft dieser Trick leider auch nicht. Auch die Angabe der Breite mit `width:100%` schafft hier keine Abhilfe (mehr zu `border` und `width` erfahren Sie in Kapitel 7).

Bei Hintergrundgrafiken zeigt Netscape 4 den Block an. Bei lokalen Stilen (durch Setzen des `style`-Attributs) werden transparente Bereiche der Grafik aber schwarz dargestellt, solange die Hintergrundfarbe nicht ebenfalls lokal gesetzt wird. Beim zweiten Absatz in Abbildung 6.5 wurde als lokale Hintergrundfarbe der gleiche Farbton angegeben wie für die Seite.

Der Grafik-Bug tritt auch auf, wenn eine teiltransparente Hintergrundgrafik mit der internen oder externen Methode definiert wird, dort aber keine Hintergrundfarbe gesetzt wird.

Hintergründe in Inline-Tags

In Inline-Tags werden Hintergrundfarben von den gängigen Browser ohne Probleme angezeigt.

Anders sieht es bei den Hintergrundgrafiken aus. Netscape 4 zeigt auch Grafiken in Inline-Tags an. Dabei wird das Inline-Tag aber wie ein Blocklevel-Tag behandelt; der nachfolgende Text beginnt in der nächsten Zeile (siehe Abbildung 6.6). In Netscape 6 ist auch dieses Problem beseitigt.

Opera zeigt in Inline-Tags die Hintergrundgrafik gar nicht an (siehe Abbildung 6.7).

IE5 hingegen zeigt die Inline-Tags so an, wie es sein soll (siehe Abbildung 6.8).

Abbildung 6.6: Anzeige im Netscape 4

Abbildung 6.7: Anzeige im Opera

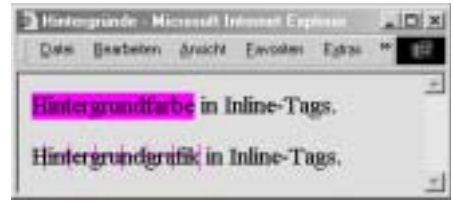

Abbildung 6.8: Anzeige im IE5

✓ Tipp:

- Vorläufig sollten Sie auf Hintergrundgrafiken in Inline-Tags verzichten. Dass Opera sie gar nicht anzeigt, wäre nicht weiter schlimm. Sehr störend wirkt sich aber die Eigenart von Netscape 4 aus, das Inline-Tag als Blocklevel-Tag zu behandeln.

```
<html><head>
<title>Hintergrundfarben</title>
<style type="text/css"><!--
body { background-color:#FFFFFF; }
h2 { background-color:#404040;
font:16pt Arial,sans-serif;
color:#E0E0E0; }
p { font:13pt Arial,sans-serif;
color:#404040; }
.intro { background-color:#E0E0E0;
font:bold 13pt Arial,sans-serif;
color:#404040; }
.neu { background-color:#E0E0E0;
font:italic bold 13pt Arial,sans-serif;
color:#000000; }
--></style></head>
<body>
<h2>Webmaster-Tutorials</h2>
<p class="intro">Auf den folgenden Seiten
finden Webmaster einige Tutorials zum
Thema Homepages.</p>
<p>Zu den Tutorials gehören unter anderem
Workshops zu HTML, CSS und DHTML. </p>
<p><span class="neu">Neu!</span> Jetzt
auch mit Tipps und Tricks zum Aufbereiten
von Grafiken. </p>
</body></html>
```

Listing 6.1: Definition von Hintergrundfarben

Abbildung 6.9: Browseranzeige

Hintergrundfarben

Die Eigenschaft zum Festlegen der Hintergrundfarbe heißt `background-color`. Als Werte sind nur Farbangaben erlaubt.

So definieren Sie eine Hintergrundfarbe:

1. `background-color:`
   Geben Sie `background-color` ein, gefolgt von einem Doppelpunkt.
2. `Farbangabe;`
   Schreiben Sie dahinter den gewünschten Farbwert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 6.1 zeigt ein Beispiel, in Abbildung 6.9 sehen Sie das Ergebnis im Browser.

✓ Tipp:

- Die verwendeten Farben sollten ein stimmiges Erscheinungsbild ergeben und müssen immer auch auf das Ziel der Seite abgestimmt werden. Eine Webseite für Kinder verträgt mehr Farbe als eine Firmenpräsenz.

Hintergrundgrafiken mit CSS

Mit dem Befehl `background-image` können Sie eine Hintergrundgrafik in ein Element einfügen. CSS beschränken sich aber nicht nur auf das Einfügen einer Grafik, Sie können ihr auch bestimmte Eigenschaften zuordnen. So kann man z.B. die Kachelung beeinflussen oder ganz unterdrücken, und die Position der Grafik festlegen.

Folgende Eigenschaften sind für eine Hintergrundgrafik möglich:

- `background-repeat` legt die Kachelung der Grafik fest.
- `background-position` legt fest, wo die obere linke Ecke der Grafik beginnt.
- `background-attachment` legt fest, ob die Grafik bei längeren Seiten mitscrollt.

Die erlaubten Werte für die Eigenschaften sehen Sie in Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3. Mehr Informationen zu den Eigenschaften finden Sie auf den nachfolgenden Seiten.

Netscape 4 unterstützt nur die Eigenschaft `background-repeat`.

Werte für background-repeat	
`repeat`	Standard; die Grafik wird gekachelt, bis die Elementbox in Höhe und Breite komplett gefüllt ist.
`repeat-x`	Die Grafik wird nur horizontal gekachelt.
`repeat-y`	Die Grafik wird nur vertikal gekachelt.
`no-repeat`	Die Grafik wird gar nicht gekachelt. Sie steht nur einmal auf der Seite.

Tabelle 6.1: Erlaubte Werte für die Eigenschaft `repeat`

Werte für background-position	
Sie können Maße und Prozentwerte sowie die folgenden Angaben verwenden:	
`top`	Oben am Element (vertikal)
`bottom`	Unten am Element (vertikal)
`left`	Links am Element (horizontal)
`right`	Rechts am Element (horizontal)
`center`	In der Mitte des Elements. Kann für die horizontale und die vertikale Ausrichtung verwendet werden.

Tabelle 6.2: Erlaubte Werte für die Eigenschaft `position`

Werte für background-attachment	
`fixed`	Standard: die Grafik wird nicht gescrollt.
`scroll`	Die Grafik wird bei Bedarf gescrollt.

Tabelle 6.3: Erlaubte Werte für die Eigenschaft `attachment`

```
<html><head>
<title>Hintergrundgrafik</title>
<style type="text/css"><!--
body { background-color:#FFFFFF; }
h2 { background-color:#404040;
font:15pt Arial,sans-serif;
color:#E0E0E0; }
p { font:13pt Arial,sans-serif;
color:#000000;
background-image:url(karo2.gif); }
--></style></head>
<body>
<h2>Webmaster-Tutorials</h2>
<p style="background-image:none;">
Auf den folgenden Seiten finden Webmaster
einige Tutorials zum Thema Homepages.</p>
<p>Zu den Tutorials gehören unter anderem
Workshops zu HTML, CSS und DHTML. <br>
 </p>
</body></html>
```

Listing 6.2: Hintergrundgrafik einfügen

Abbildung 6.10: Das Gitter ist eine Hintergrundgrafik

Eine Hintergrundgrafik einfügen

Durch CSS bieten sich Ihnen bei Hintergrundgrafiken neue Möglichkeiten. Sie sollten aber wie bei HTML darauf achten, dass der Text gut lesbar bleibt. Außerdem sollte die Dateigröße so gering wie möglich gehalten werden, damit die Ladezeit der Seite nicht allzu sehr verzögert wird.

So fügen Sie eine Hintergrundgrafik ein:

1. `background-image:`
   Geben Sie `background-image` ein, gefolgt von einem Doppelpunkt.
2. `url`
   Schreiben Sie dahinter den Wert `url`.
3. `(verzeichnis/grafik.gif);`
   Direkt im Anschluss wird in runden Klammern angegeben, welche Grafik aufgerufen wird. Achten Sie darauf, den Pfad zur Grafik richtig anzugeben. Das Semikolon schließt wie immer die Stilregel ab.

Außer `url` gibt es noch den Wert `none`. Damit kann man für einen Bereich die Hintergrundgrafik unterdrücken. Listing 6.2 zeigt ein Beispiel, in Abbildung 6.10 sehen Sie das Ergebnis im Browser.

✓ Tipp:

- Denken Sie daran, dass Netscape 4 transparente Bereiche einer Hintergrundgrafik nur dann richtig anzeigt, wenn auch eine Hintergrundfarbe angegeben wird.

Die Kachelung der Grafik

Wie schon erwähnt können Sie mit CSS über den Befehl `background-repeat` die Kachelung der Hintergrundgrafik bestimmen. Wenn Sie den Befehl weglassen, verhält sich die Hintergrundgrafik so, wie Sie das von HTML gewöhnt sind: Sie wird in der Höhe und Breite so oft wiederholt, bis der Bereich gefüllt ist. Diese vollständige Kachelung erreichen Sie auch mit dem Wert `repeat`.

Mit dem Wert `no-repeat` bestimmen Sie, dass die Grafik gar nicht wiederholt wird; sie erscheint nur ein einziges Mal auf der Seite. Der Wert `repeat-x` legt fest, dass die Grafik nur horizontal gekachelt wird, `repeat-y` bewirkt die vertikale Kachelung.

So legen Sie die Kachelung fest:

1. `background-repeat:`
   Geben Sie `background-repeat` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Ein Beispiel sehen Sie in Listing 6.3, Abbildung 6.11 zeigt das Ergebnis im Browser. Die Hintergrundgrafik des Bodys ist der Schriftzug "CSS und DHTML"; er wird nicht wiederholt. Für die Absätze wurde die Karo-Grafik mit horizontaler Kachelung gesetzt. Damit die Grafiken nicht vom Text überlappt werden, wurde der Text mit Zeilenumbrüchen nach unten gerückt. Natürlich geht das auch mit CSS, mehr dazu erfahren Sie in Kapitel 7.

```
<html><head>
<title>Hintergrundgrafik</title>
<style type="text/css"><!--
body { background-color:#FFFFFF;
background-image:url(logo.gif);
background-repeat:no-repeat; }
p { font:13pt Arial,sans-serif;
color:#000000;
background-image:url(karo.gif);
background-repeat:repeat-x; }
--></style></head>
<body><br><br><br>
<p><br><br>Auf den folgenden Seiten finden
Sie einige Tutorials zum Thema Cascading
Style Sheets (CSS).</p>
</body></html>
```

Listing 6.3: Hintergrundgrafik kacheln

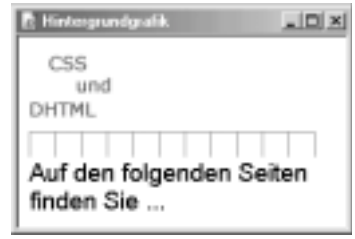

Abbildung 6.11: Browseranzeige

```
<html><head>
<title>Hintergrundgrafik</title>
<style type="text/css"><!--
body { background-color:#FFFFFF;
background-image:url(logo.gif);
background-repeat:no-repeat;
background-attachment:fixed;
padding:80px;
font:13pt Arial,sans-serif;
color:#000000; }
--></style></head>
<body><br><br>
<p><br><br>Auf den folgenden Seiten finden
Sie einige Tutorials zum Thema Cascading
Style Sheets (CSS).</p>
<p>leer</p><p>leer</p><p>leer</p>
<p>leer</p><p>leer</p><p>leer</p>
<p>leer</p><p>leer</p><p>leer</p>
<p>leer</p><p>leer</p>
</body></html>
```

Listing 6.4: Hintergrundgrafik fixieren

Abbildung 6.12: Die Hintergrundgrafik ist mit dem Wert `fixed` angegeben. Sie bleibt immer sichtbar, auch wenn die Seite gescrollt wird. Der Text wurde mit der Eigenschaft `padding` um die Breite der Grafik nach rechts gerückt.

Scrollen der Grafik

Bei längeren Seiten kann das Wegscrollen der Hintergrundgrafik verhindert werden; sie ist dann immer sichtbar. Die CSS-Eigenschaft heißt `background-attachment`. Der Wert `scroll` entspricht dem Standard, wenn Sie die Eigenschaft ganz weglassen: Die Grafik wird mit dem Seiteninhalt gescrollt und verschwindet so irgendwann aus dem Anzeigebereich. Durch den Wert `fixed` wird die Grafik nicht mitgescrollt. Optisch gesehen bleibt die Grafik dabei immer an dem vorgesehenen Platz stehen.

Das Scrollen des Hintergrunds wirkt sich nur aus, wenn die Kachelung der Grafik unterdrückt wird.

So fixieren Sie eine Grafik:

1. `background-attachment:`
   Geben Sie `background-attachment` ein, gefolgt von einem Doppelpunkt.
2. `fixed;`
   Schreiben Sie dahinter den Wert `fixed`, gefolgt von einem Semikolon, das die Stilregel abschließt.

Ein Beispiel sehen Sie in Listing 6.4, Abbildung 6.12 zeigt das Ergebnis im Browser.

✓ Tipp:

- Fixierte Hintergrundgrafiken kommen erst dann richtig zur Geltung, wenn der Text entsprechend eingerückt wird. Mehr dazu erfahren Sie in Kapitel 7.
- Netscape 4 unterstützt die Fixierung der Grafik nicht; sie wird immer mit dem Seiteninhalt mitgescrollt.

Die Position der Grafik

Wenn Sie mit `background-repeat:no-repeat` die Kachelung der Grafik unterdrücken, können Sie festlegen, wo die einzelne Grafik erscheinen soll. Die Eigenschaft heißt `background-position`.

Für die Position werden zwei Werte angegeben, die durch ein Leerzeichen voneinander getrennt werden: zuerst die horizontale Position, dann die vertikale Position. Bezugspunkt ist dabei immer die linke obere Ecke des Elements, das mit der Grafik versehen wird.

Bei Prozentangaben wird die Breite und Höhe des Elements (siehe Kapitel 7) berücksichtigt. Eine Angabe von 20% rückt die Grafik bei einem 400px breiten Element weiter nach rechts als bei einer Breite von 200px.

Bei den beschreibenden Werten stehen `left` und `right` für die horizontale Ausrichtung, `top` und `bottom` für die vertikale Ausrichtung. `center` kann für beide Richtungen verwendet werden. Das Mischen von Maß -oder Prozentangaben und beschreibenden Werten ist möglich.

`background-position:center top;` richtet die Grafik z.B. zentriert oben aus (siehe Abbildung 6.13).

`background-position:140px center;` zeigt die Grafik mit einer horizontalen Postion von 140px und vertikal zentriert an, siehe Abbildung 6.14. Die Breite des Elements wurde dabei mit 200px festgelegt (siehe Kapitel 7). Da die Grafik selbst 80px breit ist, wird sie rechts abgeschnitten.

Sie können auch negative Werte verwenden, dabei wird die Grafik ebenfalls beschnitten (siehe Abbildung 6.15).

Abbildung 6.13: center top

Abbildung 6.14: 140px center

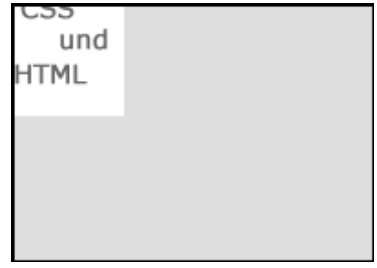

Abbildung 6.15: -20px -20px

```
<html><head>
<title>Hintergrundgrafik</title>
<style type="text/css"><!--
body { background-color:#FFFFFF;
background-image:url(logo.gif);
background-repeat:no-repeat;
background-position:right top;
font:13pt Arial,sans-serif;
color:#000000; }
--></style></head>
<body>
<p><br><br>Auf den folgenden Seiten finden
Sie einige Tutorials zum Thema Cascading
Style Sheets (CSS).</p>
</body></html>
```

Listing 6.5: Position der Hintergrundgrafik

Abbildung 6.16: Die Hintergrundgrafik ist rechts oben positioniert.

So legen Sie die Position fest:

1. `background-position:`
   Geben Sie `background-position` ein, gefolgt von einem Doppelpunkt.
2. `horizontal`
   Schreiben Sie dahinter den Wert für die horizontale Ausrichtung, gefolgt von einem Leerzeichen.
3. `vertikal;`
   Nach dem Leerzeichen notieren Sie denWert für die vertikale Ausrichtung, gefolgt von einem Semikolon, das die Stilregel abschließt.

Ein Beispiel sehen Sie in Listing 6.5, Abbildung 6.16 zeigt das Ergebnis im Browser.

Netscape 4 unterstützt die Positionierung einer Hintergrundgrafik nicht. Sie wird immer links oben angezeigt.

✓ Tipp:

- Wenn Sie eine Hintergrundgrafik beidseitig zentrieren möchten, reicht die einmalige Angabe von `center`. Der zweite `center`-Wert kann weggelassen werden.

Kurzschrift für den Hintergrund

Auch für Hintergründe gibt es eine Kurzschrift. Mit dem Befehl `background` lassen sich die Eigenschaften zusammenfassen.

So wenden Sie die Kurzschrift an:

1. `background:`
   Geben Sie die Eigenschaft `background` ein, gefolgt von einem Doppelpunkt.
2. `url(grafikdatei) Kachelung Fixierung Position Farbe;`
   Schreiben Sie dahinter die gewünschten Werte, getrennt durch Leerzeichen. Abgeschlossen wird die Stilregel wie immer durch ein Semikolon.

Nicht benötigte Werte können Sie weglassen. Listing 6.6 zeigt die Kurzschrift im Einsatz. In Abbildung 6.17 sehen Sie das Ergebnis im Browser.

✓ Tipps:

- Bei der Kurzschrift für Schrifteigenschaften spielt die Reihenfolge der Eigenschaften eine Rolle, bei der Kurzschrift für Hintergründe ist die Reihenfolge egal. Die beiden Werte für die Position müssen aber nebeneinander stehen.
- Eigenschaften, die nicht benötigt werden, können weggelassen werden.

```
<html><head>
<title>Kurzschrift</title>
<style type="text/css"><!--
body { background:url(logo.gif) no-repeat
#FFFFFF  left top scroll;
font:12pt Arial,sans-serif;
color:#000000; }
p { padding-left:30px;
padding-top:60px;}
 --></style></head>
<body>
<p>CSS eröffnen Ihnen neue Möglichkeiten
beim Seitendesign.</p>
</body></html>
```

Listing 6.6: Kurzschrift für den Hintergrund

Abbildung 6.17: Browseransicht

7 GRÖSSEN, RAHMEN, ABSTÄNDE UND RÄNDER

Wer mit HTML arbeitet, wünscht sich oft, die Größe eines Elements oder den Abstand zu anderen Abschnitten steuern zu können. Die einzige Möglichkeit, in HTML z.B. einen Absatz auf eine bestimmte Breite zu begrenzen, ist eine Tabelle mit festgelegter Pixelbreite, die den Absatz aufnimmt. Auch die Abstände lassen sich mit Tabellen in geringem Maße steuern.

Tabellen verlangsamen aber immer den Seitenaufbau, sie machen den Quelltext unübersichtlich und ein einziges fehlendes Tag kann dazu führen, dass gar nichts mehr angezeigt wird. Davon abgesehen können Tabellen die Wünsche eines Webdesigners nur begrenzt erfüllen. Bei Rahmen für Elemente stößt man dabei ebenso rasch an die Grenzen des Machbaren wie bei dem Wunsch nach unterschiedlich breiten Rändern.

CSS bieten Ihnen auch für die Größen, Rahmen, Abstände und Ränder vielfältige Möglichkeiten.

Elementgrößen

Normalerweise legt ein Browser automatisch fest, wie viel Platz ein Element benötigt. Bei der Breite richtet er sich nach der zur Verfügung stehenden Anzeigefläche, bei der Höhe wird einem Element nur so viel Platz eingeräumt, wie der Inhalt benötigt. Die Höhe passt sich automatisch an, wenn die Breite der Anzeigefläche sich ändert.

CSS lassen Ihnen die Möglichkeit offen, Breite und Höhe eines Blocklevel-Elements selbst zu bestimmen. Netscape 4 unterstützt die Höhe eines Elements leider nicht. Auch IE und Opera zeigen die Höhe eines Elements nur dann richtig an, wenn dem Browser mitgeteilt wird, wie er reagieren soll, wenn der Inhalt größer ist als das Element (siehe Kapitel 9).

Die Breite eines Blocklevel-Elements zeigen die gängigen Browser ohne Probleme an. Da sich die Höhe nach dem Inhalt und der Breite des Elements richtet, kann man mit den Einschränkungen der Browser recht gut leben, wenn man den Inhalt unter Kontrolle hält. Sie sollten also in jedem Fall die Schriftgrößen festlegen, wenn Sie bei Blocklevel-Tags mit festen Elementbreiten arbeiten möchten.

Bei Inline-Tags und Replaced-Tags bieten auch IE und Opera nur bedingt Unterstützung für die Größe an. Allerdings sind die Größen laut W3C auch nur für Blocklevel-Tags vorgesehen.

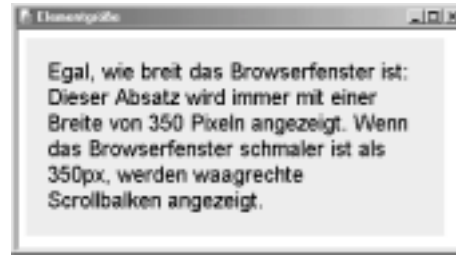

Abbildung 7.1: Wenn das Browserfenster breiter aufgezogen wird, behält der Absatz seine Breite von 350 Pixeln. Wenn das Fenster schmaler wird als das Element, werden Scrollbalken angezeigt, damit der Besucher alle Inhalte erreichen kann.

✓ Tipp:

- Aus welchen Bestandteilen sich die Elementbox zusammensetzt und dass sich die Höhe und Breite auf den Inhalt des Elements beziehen, wurde bereits im Abschnitt *Die Elementbox* in Kapitel 6 erläutert.
- Wenn Sie Elemente mit festgelegten Breiten verwenden, sollten Sie auch an die Besucher mit älteren Monitoren denken. Die Auflösung 800x600 ist immer noch als Standard anzusehen. Damit auch bei dieser Auflösung keine waagerechten Scrollbalken auftreten, sollte keines der Elemente breiter als max. 740 Pixel sein.

```
<html><head>
<title>Elementbreite</title>
<style type="text/css">
<!--
.eins { font:20px Arial,sans-serif;
background:#F0F0F0; border:none;
width:320px; }
.zwei { font:20px Arial,sans-serif;
background:#F0F0F0; border:none;
width:200px; }
--></style></head>
<body><p class="eins">Die Absätze sind
unterschiedlich breit. Dieser ist 320
Pixel breit.</p>
<p class="zwei">Dieser Absatz ist 200
Pixel breit.</p>
<p class="eins">Dieser wieder 320 Pixel.
</p></body></html>
```

Listing 7.1: Elementbreite

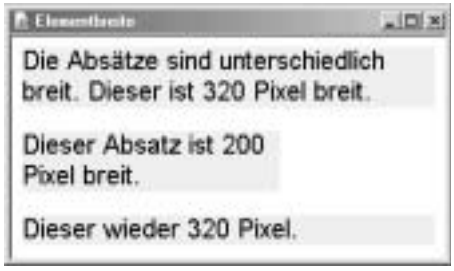

Abbildung 7.2: Browseransicht

Die Breite eines Elements

Die Eigenschaft zum Festlegen der Breite heißt `width`. Erlaubte Werte sind `auto`, *alle Maße* und *Prozentangaben*. Die Prozentwerte beziehen sich dabei immer auf das Elternelement. DerWert `auto` überlässt dem Browser die Festlegung der Breite.

So legen Sie die Breite fest:

1. `width:`
   Geben Sie die Eigenschaft `width` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

In Listing 7.1 sehen Sie ein Beispiel, Abbildung 7.2 zeigt das Ergebnis im Browser.

✓ Tipp:

- Verwenden Sie Pixelangaben für die Breite. So wird sichergestellt, dass die Seite bei möglichst vielen Besuchern so angezeigt wird, wie Sie sich das vorgestellt haben.

Die Höhe eines Elements

Die Eigenschaft zum Festlegen der Höhe heißt `height`. Erlaubte Werte sind wie bei `width` `auto`, *alle Maße* und *Prozentangaben*. Die Prozentwerte beziehen sich auch hier auf das Elternelement. Der Wert `auto` überlässt dem Browser die Festlegung der Höhe.

So legen Sie die Höhe fest:

1. `height:`
   Geben Sie die Eigenschaft `height` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

In Listing 7.2 sehen Sie ein Beispiel, Abbildung 7.3 zeigt das Ergebnis im Browser.

✓ Tipp:

- Bei Fließtexten sollten Sie die Angabe der Höhe weglassen, da Netscape 4 die Eigenschaft `height` nicht kennt.
- Wenn Sie die Höhe einsetzen, legen Sie mit dem Befehl `overflow` immer fest, was mit dem Überschuss passieren soll (siehe hierzu Kapitel 9). Fast alle Browser ignorieren die Höhenangabe, wenn `overflow` nicht definiert ist.

```
<html><head>
<title>Elementhöhe</title>
<style type="text/css">
<!--
.eins { font:20px Arial,sans-serif;
background:#F0F0F0; border:none;
width:320px;
height:80px;
overflow:auto; }
.zwei { font:20px Arial,sans-serif;
background:#F0F0F0; border:none;
width:320px;
height:40px;
overflow:auto; }
--></style></head>
<body><p class="eins">Die Absätze sind
unterschiedlich hoch. Dieser ist 80 Pixel
hoch.</p>
<p class="zwei">Dieser Absatz ist 40 Pixel
hoch.</p>
<p class="eins">Dieser wieder 80 Pixel.
</p></body></html>
```

Listing 7.2: Elementhöhe

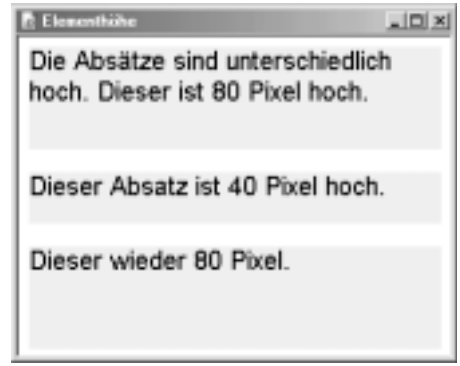

Abbildung 7.3: Elementhöhe im Browser

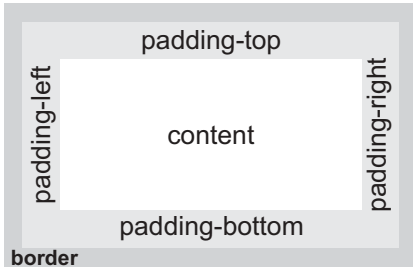

Abbildung 7.4: Die Abstände (`padding`) bezeichnen den Leerräume zwischen dem Seiteninhalt (`content`) und dem Rand (`border`) des Elements.

✓ Tipps:

- Der Abstand auf der rechten und linken Seite hängt auch von der Textausrichtung ab.
- Auch für Inline-Tags können Abstände definiert werden. Die Anzeige in den Browsern ist aber ähnlich unsauber wie bei Hintergrundgrafiken. Verzichten Sie daher besser auf Abstände in den Inline-Tags.

Abstände festlegen

Als Abstände bezeichnet man die Leerräume zwischen den Inhalten und dem Elementrahmen (siehe hierzu den Abschnitt *Die Elementbox* in Kapitel 6). Die Eigenschaft zum Festlegen der Abstände hat den engl. Begriff *padding* für die Zwischenräume übernommen. Dabei ist `padding` der Kurzschrift-Befehl für Abstände. In ihm werden die folgenden Eigenschaften zusammengefasst:

- *padding-top*
- *padding-right*
- *padding-bottom*
- *padding-left*

Die Bezeichnungen für die einzelnen `padding`-Eigenschaften ergeben sich aus den Seitenbezeichnungen des Elements (siehe Abbildung 7.4).

Bei der Kurzschrift mit `padding` haben Sie drei verschiedene Möglichkeiten:

1. Sie setzen einen Wert, der den Abstand für alle vier Seiten bestimmt.
2. Es werden zwei Werte angegeben: Der erste steht für das Paar `top`/`bottom`, der zweite für das Paar `left`/`right`.
3. Sie setzen alle vier Werte, wenn Sie unterschiedliche Abstände wünschen. Die Reihenfolge der Seiten ist dabei festgelegt: `top`, `right`, `bottom`, `left`.

Als Werte sind *Maße* und *Prozentangaben* erlaubt. Die Prozentwerte beziehen sich dabei auf die Breite des Elternelements.

So legen Sie gleichmäßige Abstände fest:

1. `padding:`
   Geben Sie die Eigenschaft `padding` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, der den Abstand für alle Seiten gleichzeitig festlegt, gefolgt von einem Semikolon, das die Stilregel abschließt.

Diese Methode setzt einen gleichmäßig breiten Abstand für die vier Seiten des Elements.

So legen Sie ungleichmäßige Abstände mit der Kurzschrift fest:

1. `padding:`
   Geben Sie die Eigenschaft `padding` ein, gefolgt von einem Doppelpunkt.
2. `oben rechts unten links;`
   Schreiben Sie dahinter die gewünschten Werte in der Reihenfolge `oben`, `rechts`, `unten` und `links`. Sie werden jeweils durch ein Leerzeichen voneinander getrennt. Nach dem Wert für den linken Abstand folgt das Semikolon, das die Stilregel abschließt.

Sie können auch die Schreibweise mit zwei Werten wählen. Die erste Angabe steht dann für das Paar `top/bottom`, die zweite für das Paar `left/right`.

Listing 7.3 zeigt ein Beispiel für beide Kurzschrift-Methoden. In Abbildung 7.5 sehen Sie das Ergebnis im Browser.

```
<html><head>
<title>Abstände</title>
<style type="text/css">
<!--
.eins { font:16px Arial,sans-serif;
background:#D0D0D0; border:none;
padding:20px;
text-align:justify; }
.zwei { font:16px Arial,sans-serif;
background:#D0D0D0; border:none;
padding:0px 10px 50px 30px;
text-align:justify; }
--></style></head>
<body><p class="eins">Dieser Absatz hat
gleichmäßige Abstände. Die Textausrichtung
wurde auf Blocksatz gesetzt.</p>
<p class="zwei">Hier sind die Abstände
ungleichmäßig breit. Auch hier wurde als
Textausrichtung der Blocksatz verwendet.</
p>
</body></html>
```

Listing 7.3: Gleichmäßige und ungleichmäßige Abstände...

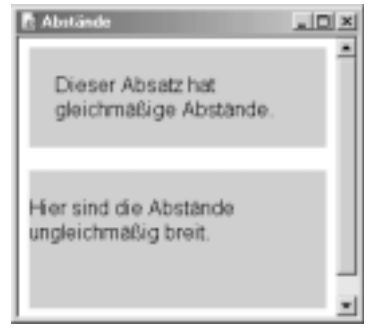

Abbildung 7.5: ...sehen im Browser so aus

```
<html><head>
<title>Abstände</title>
<style type="text/css">
<!--
p { font:16px Arial,sans-serif;
background:#D0D0D0; border:none;
padding-left:40px;
padding-top:30px;
text-align:justify; }
--></style></head>
<body><p>Die Abstände wurden mit der ausführlichen Methode festgelegt.</p>
</body></html>
```

Listing 7.4: Abstände mit ausführlicher Methode

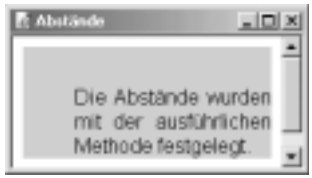

Abbildung 7.6: Browseransicht

So legen Sie einzelne Abstände fest:

1. `padding-left:`
   Geben Sie die Eigenschaft für den gewünschten Seitenabstand ein, gefolgt von einem Doppelpunkt.. Hier wurde `padding-left` für den linken Abstand gewählt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.
3. Wiederholen Sie die Schritte 1 und 2, wenn Sie für eine andere Seite ebenfalls den Abstand festlegen möchten. In Schritt 1 setzen Sie dabei jeweils die Eigenschaft für die gewünschte Seite.

Wenn Sie die Seitenabstände mit der ausführlichen Methode festlegen, ist die Reihenfolge, in der die Seiten definiert werden, egal. Sie spielt nur bei der Kurzschrift eine Rolle.

Listing 7.4 zeigt ein Beispiel für die ausführliche Methode. In Abbildung 7.6 sehen Sie das Ergebnis im Browser.

Rahmeneigenschaften

Der Rahmen ist die äußere Begrenzung des eigentlichen Elements (siehe hierzu den Abschnitt *Die Elementbox* in Kapitel 6). Rahmen gehören zu den komplexesten Eigenschaften in CSS. Grundsätzlich gibt es für Rahmen die folgenden Eigenschaften:

- `border-width` für die Rahmenbreite (Dicke)
- `border-style` für die Art der Rahmenlinie
- `border-color` für die Rahmenfarbe

So angegeben, gelten die Rahmeneigenschaften für alle vier Seiten. Sie lassen sich mit dem Kurzschrift-Befehl `border` zusammenfassen, der ebenfalls grundsätzlich für alle Seiten des Elements gilt.

Bei `padding` haben wir schon gesehen, dass man für jede Seite des Elements die Eigenschaften einzeln angeben kann. Bei den Rahmen ist das genauso. Dazu werden die einzelnen Eigenschaften durch Einfügen der Seitenbezeichnung erweitert. `border-left-width` ist z.B. die Eigenschaft für die Dicke der linken Rahmenseite, `border-bottom-style` legt die Art der Rahmenlinie für die untere Rahmenseite fest.

Für jede Seite lässt sich auch ein erweiterter Kurzschrift-Befehl verwenden. Mit `border-left` können Sie z.B. die drei Eigenschaften (Dicke, Art und Farbe) für die linke Rahmenseite zusammenfassen.

Daraus ergeben sich für Rahmen 15 einzelne Eigenschaften und 5 verschiedene Kurzschrift-Befehle (siehe Tabelle 7.1). Lassen Sie sich davon nicht abschrecken – in der Praxis sind Rahmen einfacher zu handhaben, als es sich in der Theorie anhört.

Rahmeneigenschaften

KURZSCHRIFT-BEFEHL	EIGENSCHAFT

`border` (alle vier Seiten)	`border-width`
	`border-style`
	`border-color`

`border-top` (Rahmenlinie oben)	`border-top-width`
	`border-top-style`
	`border-top-color`

`border-right` (Rahmenlinie rechts)	`border-right-width`
	`border-right-style`
	`border-right-color`

`border-bottom` (Rahmenlinie unten)	`border-bottom-width`
	`border-bottom-style`
	`border-bottom-color`

`border-left` (Rahmenlinie links)	`border-bottom-width`
	`border-bottom-style`
	`border-bottom-color`

Tabelle 7.1: Die Eigenschaften für die Rahmen mit den dazugehörigen Kurzschriftbefehlen

✓ Tipp:

- Bei Rahmen gilt in Bezug auf Inline-Tags das Gleiche wie für `padding`: Die Anzeige ist unsauber; verzichten Sie daher auf Rahmen in Inline-Tags.

Weitere Kurzschrift-Befehle für Rahmen	
KURZSCHRIFT-BEFEHL	EIGENSCHAFT
border-width	border-top-width border-right-width border-bottom-width border-left-width
border-style	border-top-style border-right-style border-bottom-style border-left-style
border-color	border-top-color border-right-color border-bottom-color border-left-color

Tabelle 7.2: Die Eigenschaften können als Kurzschrift-Befehl fungieren.

Noch mehr Rahmen-Kurzschrift ...

Angenommen, Sie möchten nur die Rahmenstärke angeben, diese soll aber für alle Seiten unterschiedlich sein: Das erreichen Sie mit keinem der bisher vorgestellten Kurzschrift-Befehle.

Bei Rahmen bieten CSS Ihnen weitere Möglichkeiten für die Kurzschrift. Die Haupt-Eigenschaften `border-width`, `border-style` und `border-color` werden selbst zu Kurzschrift-Befehlen (siehe Tabelle 7.2). Sie können eine dieser Eigenschaften angeben und dahinter in der die gewünschten Werte für die Seiten notieren. Dabei haben Sie folgende Möglichkeiten:

- *Ein Wert*: Der Wert gilt für alle vier Seiten.
- *Zwei Werte*: Der erste Wert gilt für oben und unten, der zweite Wert für links und rechts.
- *Drei Werte*: Der erste Wert gilt für oben, der zweite für links und rechts, der dritte Wert für unten.
- *Vier Werte*: Jede Seite hat ihren eigenen Wert. Es gilt die Reihenfolge `top`, `right`, `bottom`, `left`.

Zu komplex und verwirrend? Sie werden ihre Meinung sicher rasch ändern, nachdem Sie auf den folgenden Seiten die verschiedenen Möglichkeiten näher kennen gelernt haben. Wenn Sie sich an den Umgang mit Rahmen gewöhnt haben, werden Sie die vielfältigen Möglichkeiten bald zu schätzen wissen.

Erlaubte Werte für Rahmen

Für `border-color` sind Farbwerte erlaubt, für `border-width` Maße und bezeichnende Werte (siehe Tabelle 7.3).

Welche Werte für die Eigenschaft `border-style` möglich sind, sehen Sie in Abbildung 7.7. Die Werte `dotted` und `dashed` werden von Netscape 4 nicht unterstützt, er stellt beide Stile als durchgezogene Linie dar. Die Browser stellen die Rahmenstile teilweise unterschiedlich dar. In einigen Browsern treten die 3D-Effekte nur dann auf, wenn die angegebene Farbe für `border-color` sich von Schwarz deutlich abhebt.

Manche Browser, darunter auch IE, zeigen Rahmen nur an, wenn die Art der Rahmenlinie definiert wurde.

✓ Tipp:

- Auf der Internetseite zum Buch finden Sie die Ansicht der Rahmenstile in den verschiedenen Browsern unter: *http://www.css-dhtml.de*

Werte für border-width	
Maße	Maßangaben wie 3px
`thin`	dünne Linie
`medium`	mittelstarke Linie
`thick`	dicke Linie

Tabelle 7.3: Werte für `border-with`

Abbildung 7.7: Die Werte für `border-style` in der IE-Ansicht

```
<html><head>
<title>Rahmen</title>
<style type="text/css">
<!--
.eins { font:16px Arial,sans-serif;
background:#D0D0D0;
padding:10px;
border-width:2px;
border-style:solid;
border-color:#ff0000;
text-align:justify; }
.zwei { font:16px Arial,sans-serif;
background:#D0D0D0;
padding:10px;
border-top-width:4px;
border-top-style:dotted;
border-top-color:#ff0000;
text-align:justify; }
--></style></head>
<body><p class="eins">Dieser Absatz hat
einen roten Rahmen, der für alle vier Sei-
ten durchgezogen mit einer Stärke von 2
Pixeln dargestellt wird.</p>
<p class="zwei">Hier wird nur die obere
Rahmenlinie angezeigt. Sie ist rot gepunk-
tet und 4 Pixel dick.</p>
</body></html>
```

Listing 7.5: Beispiel für Rahmeneigenschaften

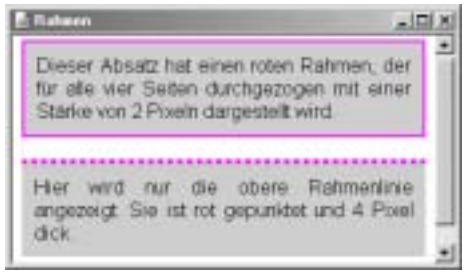

Abbildung 7.8: Ergebnis im Browser

Rahmeneigenschaften festlegen

Hier lernen Sie nun die verschiedenen Methoden näher kennen. Entscheiden Sie je nach Bedarf, welche der Möglichkeiten Sie verwenden.

Eine Eigenschaft für eine einzelne Seite festlegen:

1. `border-top-width:`
   Geben Sie die gewünschte Eigenschaft an, gefolgt von einem Doppelpunkt. Im Beispiel wurde `border-top-width` für die Stärke der linken Rahmenlinie gewählt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, der für diese Seite gelten soll, gefolgt von einem Semikolon, das die Stilregel abschließt.

Die definierte Eigenschaft gilt für eine Seite.

Eine Eigenschaft für alle Seiten festlegen:

1. `border-width:`
   Geben Sie die gewünschte Eigenschaft an, gefolgt von einem Doppelpunkt. Im Beispiel wurde `border-width` für die Rahmenstärke gewählt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, der für alle Seiten gelten soll, gefolgt von einem Semikolon, das die Stilregel abschließt.

Die Eigenschaft gilt nun für alle vier Seiten. Listing 7.5 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 7.8.

Eine Eigenschaft mit verschiedenen Werten für die vier Seiten festlegen:

1. `border-width:`
   Geben Sie die gewünschte Eigenschaft an, gefolgt von einem Doppelpunkt. Im Beispiel wurde `border-width` für die Rahmenstärke gewählt.
2. `top right bottom left;`
   Schreiben Sie dahinter die gewünschten Werte in der Reihenfolge `top`, `right`, `bottom` und `left`. Sie werden jeweils durch ein Leerzeichen voneinander getrennt. Nach dem Wert für den linken Rahmen folgt ein Semikolon, das die Stilregel abschließt.

Die Eigenschaft gilt nun mit den spezifischen Werten für die entsprechenden Seiten.

Listing 7.6 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 7.9.

✓ Tipp:

- In Schritt 2 können Sie auch die Schreibweise mit zwei oder drei Werten wählen. Bei zwei Werten gilt die erste Angabe für den Rahmen oben und unten, die zweite Angabe für den Rahmen links und rechts. Bei drei Angaben gilt der erste Wert für den Rahmen oben, der zweite für den Rahmen links und rechts. Der letzte Wert setzt den Rahmen unten.

```
<html><head>
<title>Rahmen</title>
<style type="text/css">
<!--
.eins { font:16px Arial,sans-serif;
background:#D0D0D0; padding:10px;
border-width:2px 10px 5px 1px;
border-style:dashed groove dotted solid;
border-color:#ff0000;
text-align:justify; }
.zwei { font:16px Arial,sans-serif;
background:#D0D0D0; padding:10px;
border-width:4px 10px;
border-style:solid;
border-color:#ff0000;
text-align:justify; }
--></style></head>
<body><p class="eins">Bei diesem Absatz
hat jede Rahmenlinie eine andere Stärke
und Stilart.</p>
<p class="zwei">Hier wurden für die Rahmen
zwei Werte angegeben. Der erste gilt für
die Rahmen oben/unten, der zweite für die
Rahmen links/rechts.</p>
</body></html>
```

Listing 7.6: Eigenschaften für Rahmen

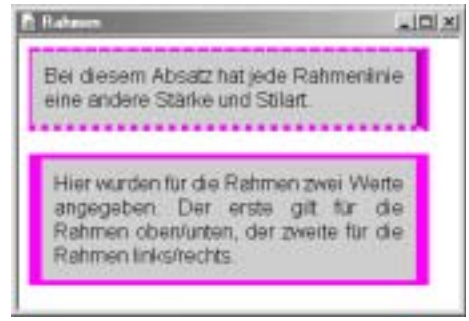

Abbildung 7.9: Ergebnis im Browser

```
<html><head>
<title>Rahmen</title>
<style type="text/css">
<!--
.eins { font:16px Arial,sans-serif;
background:#D0D0D0; padding:10px;
border-top:#ff0000 solid 3px;
text-align:justify; }
-->
</style></head>
<body><p class="eins">Bei diesem Absatz
wurden die Eigenschaften für den oberen
Rahmen durch den Kurzschrift-Befehl bor-
der-top zusammengefasst. Die Reihenfolge
der Eigenschaften ist egal.</p>
</body></html>
```

Listing 7.7: Rahmeneigenschaften für eine Seite...

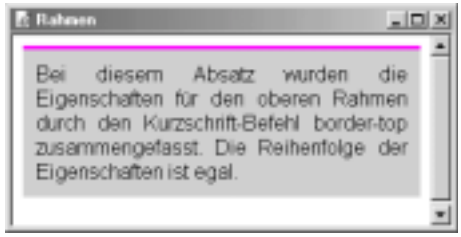

Abbildung 7.10: ...im Browser

Alle Eigenschaften für eine Seite festlegen:

Wenn nur eine Seite bestimmte Eigenschaften erhalten soll, können Sie die Kurzschrift mit der Seitenbezeichnung anwenden.

1. `border-left:`
   Geben Sie den Kurzschrift-Befehl für die gewünschte Seite an, gefolgt von einem Doppelpunkt. Hier wurde als Beispiel `border-left` für die linke Rahmenseite gewählt.
2. `width style color;`
   Schreiben Sie dahinter die gewünschten Werte für die Breite, den Linienstil und die Farbe. Sie werden jeweils durch ein Leerzeichen voneinander getrennt. Nach dem letzten Wert folgt ein Semikolon, das die Stilregel abschließt.

Die Reihenfolge der Werte ist egal. Bitte beachten Sie, dass viele Browser die Linien nur dann anzeigen, wenn auch der Linienstil gesetzt ist. Breite und Stil sollten Sie daher immer definieren, die Farbe können Sie bei Bedarf weglassen.

Listing 7.7 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 7.10.

Netscape 4 zeigt die Rahmen bei dieser Schreibweise leider nicht an.

Gleiche Eigenschaften für alle Seiten:

Wenn alle Seiten gleiche Eigenschaften haben sollen, verwenden Sie den Kurzschriftbefehl `border`.

1. `border:`
   Geben Sie den Kurzschrift-Befehl `border` an, gefolgt von einem Doppelpunkt.
2. `width style color;`
   Schreiben Sie dahinter die gewünschten Werte für die Breite, den Linienstil und die Farbe. Sie werden jeweils durch ein Leerzeichen voneinander getrennt. Nach dem letzten Wert folgt ein Semikolon, das die Stilregel abschließt.

Die Reihenfolge der Werte spielt keine Rolle. Beachten Sie, dass viele Browser die Linien nur anzeigen, wenn auch der Linienstil gesetzt ist. Breite und Stil sollten Sie daher immer definieren, die Farbe können Sie bei Bedarf weglassen.

Listing 7.8 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 7.11.

Diese Kurzschrift definiert auch Netscape 4.

```
<html><head>
<title>Rahmen</title>
<style type="text/css">
<!--
.eins { font:16px Arial,sans-serif;
background:#D0D0D0; padding:10px;
border:#ff0000 solid 3px;
text-align:justify; }
-->
</style></head>
<body><p class="eins">Bei diesem Absatz
wurden die Eigenschaften für alle Rahmen-
seiten durch den Kurzschrift-Befehl border
zusammengefasst. Die Reihenfolge der
Eigenschaften ist egal.</p>
</body></html>
```

Listing 7.8: Gleiche Eigenschaften für alle Seiten

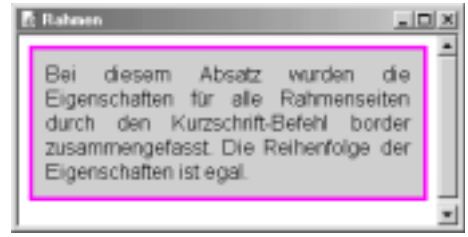

Abbildung 7.11: Browseransicht

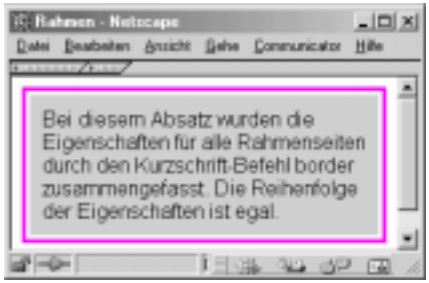

Abbildung 7.12: Netscape 4 zeigt zwischen der Hintergrundfarbe und dem Rahmen des Elements einen schmalen Streifen an. Er hat die Farbe des Seitenhintergrundes, in diesem Fall Weiß.

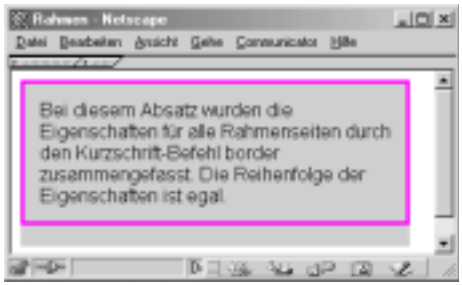

Abbildung 7.13: Mit `layer-background-color` wird der Streifen unterdrückt. Nun läuft aber bei Netscape 4 die Farbe über den Rahmen des Elements hinaus.

Netscape 4 und Rahmen

Wenn Sie Rahmen und Hintergrundfarben zusammen verwenden, tritt ein weiterer Bug in Netscape4 auf. Zwischen Hintergrundfarbe und Rahmen zeigt Netscape einen schmalen Streifen in der Farbe des Seitenhintergrundes an (siehe Abbildung 7.12).

Die einzige Möglichkeit, mit der Sie dieses Verhalten unterdrücken können, ist die Eigenschaft `layer-background-color`. Sie wirkt aber nur, wenn das Element positioniert wurde (siehe Kapitel 8) und führt zusätzlich dazu leider ebenfalls zu einem Bug. Die Farbe läuft über den Elementrahmen hinaus (siehe Abbildung 7.13).

Es bleibt natürlich Ihnen überlassen, welchen der beiden Bugs Sie in Kauf nehmen. Ich persönlich finde den schmalen Streifen zwischen Rahmen und Hintergrundfarbe akzeptabler als das Überlaufen der Farbe.

Netscape 4 und Tabellenrahmen

So frustrierend es für einen Webdesigner auch sein mag: In Tabellen erlaubt Netscape4 keine Rahmenformatierungen. Ebensowenig können Sie Breite und Höhe in Tabellen-Tags mit CSS formatieren. Dies betrifft alle Tabellen-Tags, also `<table>`, `<tr>`, `<th>` und `<td>`.

Hintergründe in Tabellen können in Netscape 4 ebenfalls Probleme machen. Bei der Formatierung von Tabellen sollten Sie daher stets mit Netscape 4 das Ergebnis überprüfen.

Ränder vs. Abstände

Ränder und Abstände werden oft verwechselt, weil sie scheinbar das Gleiche bewirken: Sie fügen Abstand um das Element ein. Bei näherer Betrachtung wird aber schnell klar, dass Ränder und Abstände unterschiedliche Zwecke erfüllen.

Während Abstände (`padding`) dafür sorgen, dass zwischen dem Inhalt und dem Rahmen des Elements ein Leerraum geschaffen wird, sind Ränder dafür zuständig, Leerräume zwischen den Elementen anzuzeigen. Die CSS-Eigenschaft für Ränder heißt `margin`.

In Abbildung 7.14 und Abbildung 7.15 sehen Sie den Unterschied zwischen Abständen und Rändern.

`margin` ist der Kurzschrift-Befehl für die Ränder; mit ihm lassen sich die folgenden Eigenschaften zusammenfassen:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

Die Bezeichnungen für die einzelnen `margin`-Eigenschaften ergeben sich wieder aus den Seitenbezeichnungen des Elements.

Erlaubte Werte sind *Maße* und *Prozentangaben*. Letztere beziehen sich auf die Breite des Elternelements. Ebenfalls erlaubt ist der Wert `auto`, was dem Standard ohne Randangabe entspricht.

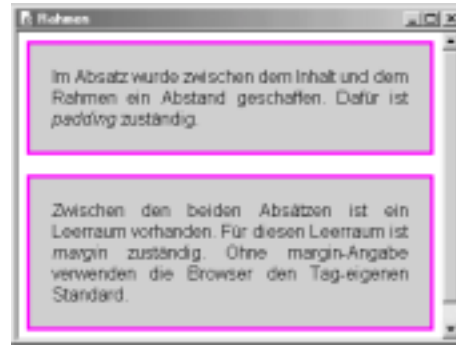

Abbildung 7.14: `padding` sorgt für den Abstand zwischen Inhalt und Rahmen des Elements.

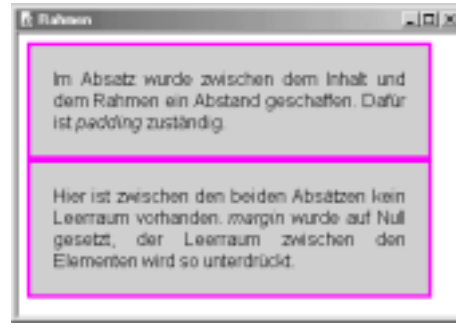

Abbildung 7.15: `margin` legt den Abstand zwischen Elementen fest. Wenn `margin` auf Null gesetzt wird, wird der Leerraum zwischen den Elementen unterdrückt.

```
<html><head>
<title>Ränder</title>
<style type="text/css">
<!--
body { background:#ffffff;
font:16px Arial,sans-serif;
margin:50px 30px 60px 40px; }
p { font:16px Arial,sans-serif;
background:#D0D0D0; border:none;
margin:30px; padding:10px;
text-align:justify; }
--></style></head>
<body>
<p>Der Seitenkörper hat ungleichmäßig
breite Ränder: Oben 50px, rechts 30px,
unten 60px, links 40px. </p>
<p>Die Absätze hingegen haben gleichmäßig
breite Ränder.</p>
</body></html>
```

Listing 7.9: Ränder festlegen

Abbildung 7.16: Ansicht im Browser

Ränder festlegen

Wie bei `padding` haben Sie auch bei `margin` für die Kurzschrift drei Möglichkeiten: ein Wert für gleichmäßige Ränder, zwei Werte für die Paare `top/bottom` und `left/right` sowie vier Werte, wenn für jede Seite ein anderer Rand gesetzt werden soll.

So legen Sie gleichmäßige Ränder fest:

1. `margin:`
   Geben Sie die Eigenschaft `margin` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, der den Rand für alle Seiten gleichzeitig festlegt, gefolgt von einem Semikolon, das die Stilregel abschließt.

Diese Methode setzt einen gleichmäßig breiten Rand für die vier Seiten des Elements.

So legen Sie ungleichmäßige Ränder mit der Kurzschrift fest:

1. `margin:`
   Geben Sie die Eigenschaft `margin` ein, gefolgt von einem Doppelpunkt.
2. `top right bottom left;`
   Schreiben Sie dahinter die gewünschten Werte in der Reihenfolge `top`, `right`, `bottom` und `left`. Sie werden jeweils durch ein Leerzeichen voneinander getrennt. Nach dem Wert für den linken Rand folgt ein Semikolon, das die Stilregel abschließt.

Sie können auch die Schreibweise mit zwei Werten für `top/bottom` und `left/right` wählen.

Listing 7.9 zeigt ein Beispiel für beide Kurzschrift-Methoden. In Abbildung 7.16 sehen Sie das Ergebnis im Browser.

So legen Sie einzelne Ränder fest:

1. `margin-left:`
   Geben Sie die Eigenschaft für den gewünschten Seitenrand ein, gefolgt von einem Doppelpunkt. Hier wurde `margin-left` für den linken Rand gewählt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.
3. Wiederholen Sie die Schritte 1 und 2, wenn Sie für eine andere Seite ebenfalls den Rand festlegen möchten. In Schritt 1 setzen Sie dabei jeweils die Eigenschaft für die gewünschte Seite.

Wenn Sie die Seitenränder mit der ausführlichen Methode festlegen, ist die Reihenfolge, in der die Seiten definiert werden, egal. Sie spielt nur bei der Kurzschrift eine Rolle.

Listing 7.10 zeigt ein Beispiel für die ausführliche Methode. In Abbildung 7.17 sehen Sie das Ergebnis im Browser.

Wenn Sie einen Wert nicht definieren, setzen die Browser den Standard, der für das Element gültig ist. Daher wird in Abbildung 7.17 zwischen den Absätzen ein Abstand angezeigt. Wir haben den oberen Rand zwar auf Null gesetzt, der untere Rand ist aber nicht definiert, also verwendet der Browser den bei Absätzen üblichen Abstand.

```
<html><head>
<title>Ränder</title>
<style type="text/css">
<!--
p { font:16px Arial,sans-serif;
background:#D0D0D0; border:none;
margin-left:20px;
margin-top: 0px;
text-align:justify; }
--></style></head>
<body>
<p>Die Abstände wurden mit der ausführlichen Methode festgelegt.</p>
<p>Noch ein Absatz.</p>
</body></html>
```

Listing 7.10: Ausfühliche Methode für Ränder...

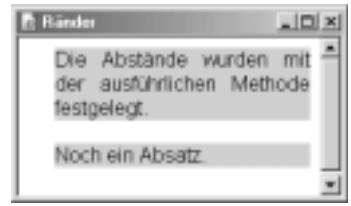

Abbildung 7.17: ...sieht im Browser so aus

```
<html><head>
<title>Textumfluss</title>
<style type="text/css">
<!--
.eins { font:16px Arial,sans-serif;
text-align:justify; color:#ff0000;
padding:15px; width:130px;
float:left; }
p { font:16px Arial,sans-serif;
background:#F0F0F0; margin:20px;
text-align:justify; padding:20px; }
--></style></head>
<body>
<p><span class="eins">Dieser Text wird vom
übrigen Text umflossen. </span> Dieser
Text umfließt den roten Text links. Mehr
Text, damit man den Umfluss auch erkennen
kann. Noch mehr Text, und noch mehr, noch
mehr,... Jetzt ist der Textumfluss ganz
deutlich zu sehen. </p></body></html>
```

Listing 7.11: Textfluss bestimmen

Abbildung 7.18: Umflossener Text im Browser

Textumfluss um ein Element

In HTML haben Sie die Möglichkeit, Text um eine Grafik fließen zu lassen. Mit CSS können Sie den Text nicht nur um Grafiken fließen lassen, sondern auch um andere Textelemente und Tabellen. Die Eigenschaft für den Textumfluss heißt `float`, erlaubte Werte sind `left`, `right` und `none`. Die Eigenschaft float wird dem Element zugewiesen, um das der Text herum fließen soll.

So bestimmen Sie den Textumfluss:

1. `float:`
   Geben Sie die Eigenschaft `float` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Besonders geeignet ist `float` natürlich, um Text um Grafiken fließen zu lassen. In Listing 7.11 wurde aber keine Grafik gewählt, sondern Text, der umflossen wird. Das Ergebnis sehen Sie in Abbildung 7.18.

✓ Tipp:

- Wenn Sie `float` verwenden, sollten Sie immer auch die Breite des zu umfließenden Bereiches mit `width` definieren.

Textumfluss verhindern

Natürlich bieten CSS auch die Möglichkeit, den Textumfluss zu verhindern bzw. abzubrechen. Die Eigenschaft heißt `clear`, erlaubte Werte sind `left`, `right`, `both` und `none`.

Der Wert `none` kann eingesetzt werden, wenn Sie z.B. für das `<p>`-Tag festgelegt haben, dass es andere Elemente nie umfließen soll, für einen bestimmten Absatz den Textumfluss jedoch erzwingen wollen.

`left` und `right` unterbrechen den Textfluss, wenn ein Element mit dem entsprechenden Richtungswert umflossen wird, beim Wert `both` wird der Textumfluss auf jeden Fall verhindert.

So verhindern Sie den Textumfluss:

1. `clear:`
   Geben Sie die Eigenschaft `clear` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

In Listing 7.12 wurde für das `<p>`-Tag mit `clear:both;` festgelegt, dass es andere Elemente nie umfließen soll. Durch eine Klasse mit der Definition `clear:none;` kann für einen Absatz der Textumfluss trotzdem erzwungen werden. Das Ergebnis sehen Sie in Abbildung 7.19.

```
<html><head>
<title>Textumfluss</title>
<style type="text/css"><!--
.eins { font: 20px Arial,sans-serif;
color:#ff0000; width:150px;
float:left; padding:5px; }
.zwei { font:16px Arial,sans-serif;
text-align:justify; border:none;
clear:none; }
p { font:16px Arial,sans-serif;
text-align:justify;
clear:both; }
--></style></head><body>
<h2 class="eins">Überschrift</h2>
<p class="zwei">Dieser Absatz umfließt die
Überschrift, weil mit clear:none; angege-
ben wurde, dass der Textumfluss erlaubt
ist.</p>
<h2 class="eins">Überschrift</h2>
<p>Dieser Absatz umfließt die Überschrift
nicht, er fängt unterhalb an, weil mit
clear:both; der Umfluss in jedem Fall ver-
hindert wird.</p>
</body></html>
```

Listing 7.12: Textumfluss verhindern

Abbildung 7.19: Browseransicht

8

ELEMENTE POSITIONIEREN

In HTML bleibt Ihnen zum Positionieren von Elementen nur die Möglichkeit, mit blinden Tabellen zu arbeiten.

Aber wie schon zu Beginn des letzten Kapitels erwähnt, verlangsamen Tabellen den Seitenaufbau und machen den Quelltext unübersichtlich.

Nichts gegen Tabellen, sie sind oft ganz nützlich, aber zum Positionieren von Elementen bieten CSS Ihnen Möglichkeiten, die Tabellen meist überflüssig machen.

CSS gehen sogar noch weiter. Elemente zu überlappen, das schaffen Sie mit HTML nicht. Mit CSS hingegen ist auch das kein Problem.

Fenster-Grundlagen

Um Elemente auf einer Seite richtig positionieren zu können, müssen Sie zuerst einmal wissen, was genau das Browserfenster ist und was ist mit dem Begriff window (engl. für Fenster) gemeint ist.

Das Browserfenster umfasst alles, was Sie sehen, wenn Sie den Browser öffnen. Dazu gehören außer der Anzeigefläche für die Webseiten auch die Adresszeile, die Buttons, die Statuszeile usw. Dieses Browserfenster müsste eigentlich als *window* bezeichnet werden, tatsächlich wird in diesem Buch aber der Begriff *Browserfenster* oder *Browserwindow* verwendet, um Verwechslungen zu vermeiden.

Mit *window* wird nämlich im Fachjargon auch die *Anzeigefläche* des Browsers bezeichnet. Diese Anzeigefläche ist das oberste Objekt einer Webseite. Es ist also gewissermaßen das Elternelement für alle anderen Elemente. Auf Webseiten wird dieses Element durch das `<body>`-Tag gebildet.

Der Startpunkt (*Ursprung*) des Anzeigefensters ist die linke, obere Ecke (siehe Abbildung 8.1). Er wird mit den Werten `0,0` angegeben (Null Pixel in beide Richtungen). An diesem Ursprung orientieren sich die Browser, wenn Elemente direkt (nicht in andere Elemente eingebettet) absolut positioniert werden und das Fenster nicht aufgeteilt wurde.

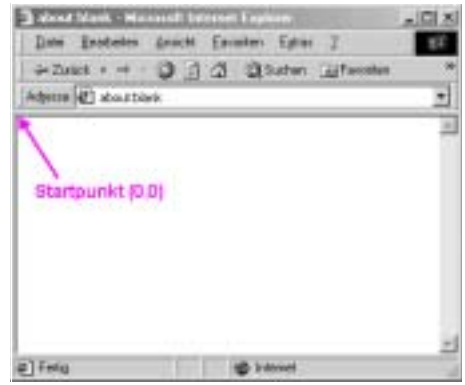

Abbildung 8.1: Die linke obere Ecke ist der Startpunkt, auf den sich Positionierungen beziehen. Man nennt diesen Punkt auch Ursprung.

✓ Tipp:

- Das Anzeigefenster kann in so genannte *Frames* (Rahmen) unterteilt werden. Mit Frames können mehrere Dateien gleichzeitig im Browser angezeigt werden. Bildlich kann man Frames mit einem Sprossenfenster vergleichen, bei dem in jede Glasscheibe ein anderes Bild (eine andere Datei) gesetzt wird. Der Ursprung eines Frames ist immer die linke obere Ecke des entsprechenden Frames.

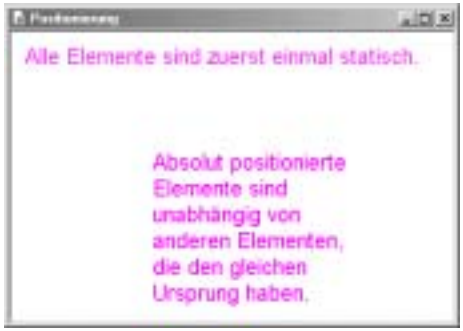

Abbildung 8.2: Der graue Schriftzug ist ein statisches Element; der rote Textblock ist absolut positioniert.

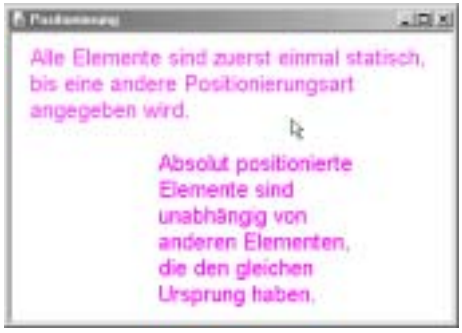

Abbildung 8.3: Auch wenn der Inhalt der anderen Seitenelemente sich ändert, wird der absolut positionierte Textblock nicht nach unten verschoben.

Arten der Positionierung

Bei Positionierungen unterscheidet man verschiedene Arten: statisch, absolut, relativ und fixiert. Die Unterschiede müssen Ihnen bekannt sein, damit Sie Elemente richtig positionieren können.

- *static*: Standardmäßig ist jedes Element, das auf der Seite definiert wird, statisch positioniert, bis eine andere Positionierungsart (relative, absolute oder fixed) festgelegt wird. Man kann die Positionierung auch explizit auf `static` festlegen, das macht aber wenig Sinn, weil die Position selbst nicht angegeben werden kann. Ein statisches Element kann mithilfe von JavaScript nicht bewegt werden.
- *absolute*: Bei der absoluten Positionierung wird ein Element an einem Ursprung verankert und ist von anderen Elementen, die den gleichen Ursprung haben, unabhängig. Wenn das Element nicht in ein anderes eingebettet ist, wird als Bezugspunkt der Ursprung von `window` verwendet. Wenn ein Element in einem anderen Element eingebettet ist, richtet es sich nach dem Startpunkt seines Elternelements. Es ist aber von allen anderen Seitenelementen unabhängig, die mit ihm zusammen im Elternelement stehen. *Beispiel:* Nach einem dreizeiligen Absatz folgt ein absolut positioniertes Element. Wenn sich der Inhalt des Absatzes ändert und er dadurch fünfzeilig wird, wird das absolut positionierte Element nicht nach unten verschoben (siehe Abbildung 8.2 und Abbildung 8.3). Ein absolut positioniertes Element kann mithilfe von JavaScript bewegt werden.

- *relative*: Bei der relativen Positionierung ist ein Element abhängig von anderen Seitenelementen. Der Bezugspunkt ist dabei sein eigener Ausgangspunkt.
  Beispiel: Auf der Seite ist ein dreizeiliger Absatz vorhanden, dahinter folgt ein relativ positioniertes Element. Wenn der Inhalt des Absatzes sich ändert und der Absatz dann mehr Zeilen enthält, verschiebt sich das nachfolgende relativ positionierte Element automatisch nach unten (siehe hierzu Abbildung 8.4 und Abbildung 8.5). Ein relativ positioniertes Element kann mithilfe von JavaScript bewegt werden.
- *fixed*: Bei der fixierten Positionierung wird das Element wie bei der absoluten Positionierung unabhängig von anderen Elementen ausgerichtet. Bei der absoluten Positionierung wird ein Element bei langen Seiteninhalten aber aus dem Anzeigebereich gescrollt, das fixierte Element bleibt hingegen auch beim Scrollen an seinem Platz. Der Effekt ist also der gleiche wie der, der bei Hintergrundgrafiken mit dem Befehl `background-attachment:fixed;` erreicht wird (siehe Kapitel 6). Fixierte Elemente können ebenfalls mit JavaScript bewegt werden. Der Traum eines Webdesigners? Ja! *Aber:* `fixed` wird bisher von *keinem* der gängigen Browser unterstützt.

`static` entspricht dem Standard ohne Positionierung, `fixed` kann noch nicht eingesetzt werden. Es bleiben also zwei Arten der Positionierung übrig, mit denen in der Praxis gearbeitet wird: relative und absolute Positionierung.

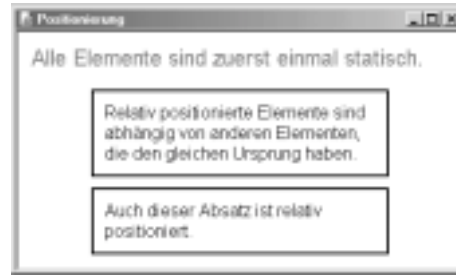

Abbildung 8.4: Der graue Schriftzug ist ein statisches Element; die beiden umrahmten Absätze sind relativ positioniert.

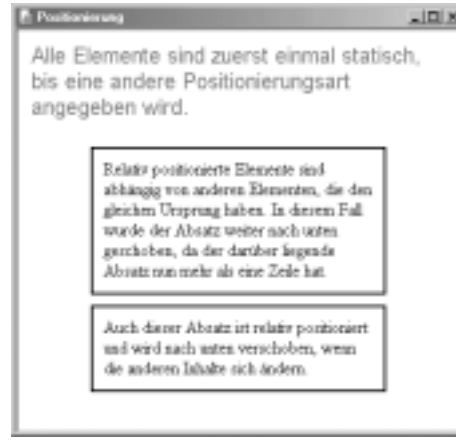

Abbildung 8.5: Die relativ positionierten Elemente werden verschoben, wenn sich die anderen Seiteninhalte ändern.

```
<html><head>
<title>Positionierung</title>
<style type="text/css">
<!--
p { font:16px Arial,sans-serif;
margin:20px; padding:10px; }
h2 { font:20px Arial,sans-serif;
background:#F0F0F0;  margin:20px;
padding:10px; }
#intro { position:absolute; }
#logo { position:relative; }
--></style></head>
<body>
<div id="intro">
<h2>Herzlich Willkommen!</h2>
<p>Wir freuen uns, dass Sie den Weg zu
<img src="logo.gif" id="logo"> gefunden
haben.</p>
</div></body></html>
```

Listing 8.1: Positionierung festlegen

✓ Tipp:

- Ein positioniertes Element wird im Fachjargon häufig einfach *absolutes Element* (für absolut positioniert) oder *relatives Element* (für relativ positioniert) genannt. Wenn also im weiteren Verlauf des Buchs von einem relativen Element die Rede ist, ist damit ein relativ positioniertes Element gemeint.

Die Positionierungsart festlegen

Die Art der Positionierung wird mit dem Befehl `position` festgelegt. Erlaubte Werte sind `absolute`, `relativ`, `fixed` und `static`. Warum die beiden Letzteren in der Praxis nicht zum Einsatz kommen, wurde bereits im letzten Abschnitt erläutert.

Ob Sie die relative oder die absolute Positionierung wählen, hängt davon ab, was Sie erreichen wollen. Elemente, die mit anderen Elementen einen zusammenhängenden Inhalt bilden (z.B. Bilder mit Text), sollten immer relativ positioniert werden. So ist sichergestellt, dass das positionierte Element bei Änderungen seinen Platz innerhalb der Inhalte behält.

So legen Sie die Positionierungsart fest:

1. `position:`
   Geben Sie die Eigenschaft `position` ein, gefolgt von einem Doppelpunkt.
2. `relative;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

In Listing 8.1 sehen Sie ein Beispiel. Die Angabe von `position` legt noch nicht den Startpunkt für ein Element fest; dazu braucht man eine Startposition (mehr dazu im nächsten Abschnitt).

Die Position oben links festlegen

Standardmäßig wird die Position eines Elements vom Bezugspunkt aus (oben links) festgelegt.

Bei der absoluten Positionierung kann der Bezugspunkt der `window`-Ursprung sein oder die linke obere Ecke des Elternelements, in das das positionierte Element eingebettet wird. Bei der relativen Positionierung ist der Bezugspunkt des Elements seine eigene linke obere Ecke.

Die Startpositionen heißen `top` (oben) und `left` (links). Erlaubte Werte sind Maße und Prozentangaben, wobei sich die Prozentwerte auf die Breite des Elternelements beziehen.

So legen Sie die Position von oben links aus fest:

1. `position:absolute;`
   Geben Sie die gewünschte Positionierungsart an.
2. `top:`
   In der nächsten Zeile schreiben Sie die Startposition `top`, gefolgt von einem Doppelpunkt.
3. `20px;`
   Geben Sie den gewünschten Wert an, gefolgt von einem Semikolon, das die Definition für `top` abschließt. Das Element wird, vom Bezugspunkt aus gesehen, um diesen Wert nach unten gerückt. Negative Werte rücken das Element nach oben.

Fortsetzung auf der nächsten Seite

✓ Tipps:

- Für die Startpositionen ist auch der Wert `auto` erlaubt. Bei absoluten Elementen legt der Browser den Startwert dann selbst fest, bei relativen Elementen ist der `auto`-Startwert immer Null. Die Angabe von `auto` macht also selten Sinn.
- Sie müssen nicht beide Startpositionen angeben; zwingend erforderlich ist aber, dass die Positionierungsart und zumindest eine der beiden Startpositionen in der gleichen Stilregel festgelegt wird.
- Für Positionierungen werden oft die Tags `<div>` und `<span>` mit IDs verwendet (siehe Kapitel 2). Einige Browser haben massive Probleme mit der Positionierung, wenn sie in anderen Tags angewendet wird. Gewöhnen Sie sich daher an, um die zu positionierenden Inhalte ein `<div>`-Tag zu legen, wenn es sich um ein absolutes Element oder einen Abschnitt handelt. Das `<span>`-Tag wird für Inline-Elemente und relative Elemente verwendet.

```
<html><head>
<title>Positionierung</title>
<style type="text/css"><!--
p { font:16px Arial,sans-serif;margin:5px;
}
h2 { font:20px Arial,sans-serif; pad-
ding:5px; background:#F0F0F0; border:none;
text-align:center;}
#intro { position:absolute; top:20px;
left:20px; width:280px; padding:0px; mar-
gin:0px;}
#logo { position:relative;
top:0px; left:100px; margin:0px; }
div { border:2px solid; }
--></style></head>

<body><div id="intro">
<h2>Herzlich Willkommen!</h2>
<p align="center">Auf dieser Homepage geht
es um </p>
<span id="logo"><img src="logo.gif"
width="80" height="80" border="0">
</span></div></body></html>
```

Listing 8.2: Positionierung von oben links aus festlegen

Abbildung 8.6: Browseransicht

4. `left:`
   In der nächsten Zeile schreiben Sie die Startposition `left`, gefolgt von einem Doppelpunkt.
5. `20px;`
   Geben Sie den gewünschten Wert an, gefolgt von einem Semikolon, das die Definition für `left` abschließt. Das Element wird, vom Bezugspunkt aus gesehen, um diesen Wert nach rechts gerückt. Negative Werte rücken das Element nach links.

In Listing 8.2 sehen Sie ein Beispiel, das Ergebnis im Browser zeigt die Abbildung 8.6.

✓ Tipp:

- Die ID für die Grafik könnte man auch direkt in das `<img>`-Tag setzen, Netscape 4 zeigt dann aber die Positionierung nicht richtig an. Wenn um die Grafik das `<span>`-Tag gelegt wird, zeigt auch N4 die Positionierung richtig an.

Die Position unten rechts festlegen

Manchmal kann es wünschenswert sein, die Position eines Elements nicht von oben links, sondern von rechts unten aus festzulegen. Ich persönlich wende diese Positionierungsart nicht an, die Gründe dafür finden Sie in den Tipps rechts. Dennoch möchte ich Ihnen diese Möglichkeit nicht vorenthalten.

So legen Sie die Position von rechts unten aus fest:

1. `position:absolute;`
   Geben Sie die gewünschte Positionierungsart an.
2. `right:`
   In der nächsten Zeile schreiben Sie die Startposition `right`, gefolgt von einem Doppelpunkt.
3. `20px;`
   Geben Sie den gewünschten Wert an, gefolgt von einem Semikolon, das die Definition für `right` abschließt. Das Element wird, vom Bezugspunkt aus gesehen, um diesen Wert nach links gerückt. Negative Werte rücken das Element nach rechts.

Fortsetzung auf der nächsten Seite

✓ Tipps:

- Die 4er-Versionen von Internet Explorer und Netscape kennen diese Art der Positionierung nicht.
- Wenn ein Element mit `top/left` positioniert wird und länger oder breiter als der zur Verfügung stehende Anzeigebereich ist, zeigen die Browser Scrollbalken an, damit der Besucher auch den Rest des Elements sehen kann. Bei der Positionierung mit `right/bottom` wird ein Element, das größer als die Anzeigefläche ist, nach oben bzw. links über die `window`-Ränder hinaus verschoben. Die meisten Browser zeigen in diesem Fall keine Scrollbalken an, der Besucher sieht die "überhängenden" Inhalte nicht (siehe Abbildung 8.7)!
- Wenn Sie für die Positionierung `top/left` und `right/bottom` verwenden, kommt es natürlich zu Konflikten. Wie das Element angezeigt wird, ist immer vom Browser abhängig, die meisten verwenden die Werte der `top/left`-Position. Sie sollten solche Konflikte aber in jedem Fall vermeiden!

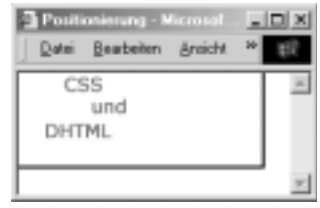

Abbildung 8.7: Die Inhalte sind links und oben abgeschnitten, weil die Positionierung mit `right/bottom` erfolgte und die Anzeigefläche nicht groß genug ist. Es werden keine Scrollbalken angezeigt.

```
<html><head>
<title>Positionierung</title>
<style type="text/css"><!--
p { font:16px Arial,sans-serif;margin:5px;
}
h2 { font:20px Arial,sans-serif; pad-
ding:5px; background:#F0F0F0; border:none;
text-align:center;}
#intro { position:absolute; right:20px;
bottom:20px; width:280px; padding:0px;
margin:0px;}
#logo { position:relative;
right:-100px; bottom:0px;
margin:0px; }
div { border:2px solid; }
--></style></head>
<body><div id="intro">
<h2>Herzlich Willkommen!</h2>
<p align="center">Auf dieser Homepage geht
es um </p>
<span id="logo"><img src="logo.gif"
width="80" height="80" border="0">
</span></div>
</body></html>
```

Listing 8.3: Position von rechts unten aus festlegen

Abbildung 8.8: Die Elemente sind mit `right/bottom` positioniert.

4. `bottom:`
   In der nächsten Zeile schreiben Sie die Startposition `bottom`, gefolgt von einem Doppelpunkt.
5. `20px;`
   Geben Sie den gewünschten Wert an, gefolgt von einem Semikolon, das die Definition für `bottom` abschließt. Das Element wird vom Bezugspunkt aus gesehen um diesen Wert nach oben gerückt. Negative Werte rücken das Element nach unten.

In Listing 8.3 sehen Sie ein Beispiel, das Ergebnis im Browser zeigt die Abbildung 8.6.

Die dritte Dimension

Ein Monitor und damit auch das Browserfenster kann nur zwei Dimensionen darstellen. Man kann aber dennoch einen dreidimensionalen Eindruck erwecken, indem Objekte einander überlappen (siehe Abbildung 8.9).

Dabei werden die einzelnen Objekte in Lagen so übereinander gestapelt, als ob teilbedruckte Klarsichtfolien aufeinander gelegt werden. Diese "Folien" oder Lagen nennt man in Grafik- und Bildbearbeitungsprogrammen wie z.B. Adobe PhotoShop Ebenen. Der Begriff Ebenen wird auch in CSS verwendet.

Mit reinem HTML haben Sie keine Chance, diesen Effekt zu erreichen, denn es gibt keine Tags, die das Stapeln von Elementen erlauben.

Mit CSS hingegen funktioniert diese Stapelung, sobald die Elemente positioniert wurden (siehe Abbildung 8.10). Das Stapeln von Ebenen wird häufig als *3D-Positionierung* bezeichnet.

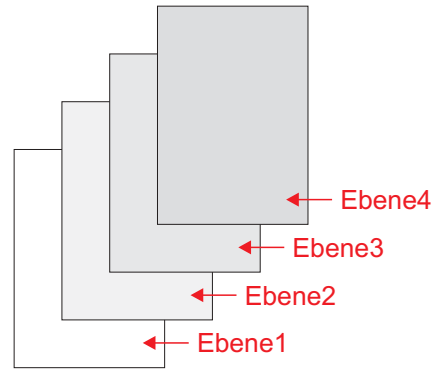

Abbildung 8.9: Ebenen können einander überlappen oder völlig überdecken. Dadurch wird ein dreidimensionaler Eindruck erweckt.

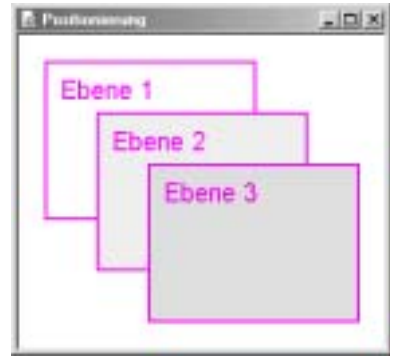

Abbildung 8.10: 3D-Positionierung im Browser

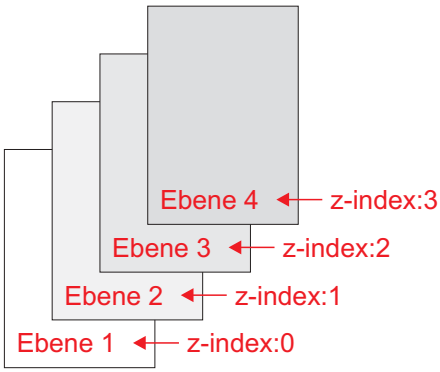

Abbildung 8.11: Die Ebenen mit den dazugehörigen z-index-Nummern

```
<html><head><style type="text/css">
<!--
#hell {position:absolute;
top:20px; left:20px;
background:#F0F0F0; }
#mittel {position:absolute;
top:50px; left:50px;
background:#D0D0D0; }
#dunkel {position:absolute;
top:80px; left:80px;
background:#B0B0B0; }
div { width:100px; height:50px;
border:2px solid; padding:5px;}
--></style></head><body>

<div id="hell">Ebene 1</div> <== z-index:0

<div id="mittel">Ebene 2</div> <== z-index:1

<div id="dunkel">Ebene 3</div> <== z-index:2

</body></html>
```

Abbildung 8.12: Den natürlichen z-index kann man durch die Reihenfolge der Elemente im Quelltext feststellen.

Reihenfolge der Elemente

Grundlage für das Stapeln und Überlappen von positionierten Elementen ist ein System, das den Namen z-index trägt.

Jedes Element, das auf der Webseite positioniert wird, erhält automatisch eine z-index-Nummer (kurz: z-index). Diese Nummern beginnen mit 0 und werden in Einserschritten erhöht, also 0, 1, 2, 3, 4 usw. In vielen Dokumentationen wird die automatische Vergabe der z-index-Nummern auch als natürlicher z-index bezeichnet.

Das Element mit dem niedrigsten z-index wird als unterstes angezeigt, liegt also unter allen anderen. Das Element mit der höchsten Nummer wird als oberstes angezeigt, und liegt dadurch über den anderen (siehe Abbildung 8.11).

Bei der automatischen Vergabe erhalten die positionierten Elemente die z-index-Nummer in der Reihenfolge, in der die Elemente im HTML-Code notiert werden.

Von den Nummern des natürlichen z-indexes sehen Sie im Quelltext nichts. Anhand der Reihenfolge, wie die positionierten Elemente im Quelltext stehen, kann man aber den natürlichen z-index erkennen (siehe Abbildung 8.12).

Den z-index festlegen

Die Stapelreihenfolge kann mit der Eigenschaft `z-index` beeinflusst werden. Als Werte sind nur Ganzzahlen erlaubt. Der z-index muss zusammen mit der Positionierungsart und der Startposition in einer Stilregel notiert werden.

So legen Sie den z-index fest:

1. `position:absolute;`
   Legen Sie die Positionierungsart fest.
2. `top:10px;`
   Geben Sie mindestens eine Startposition an.
3. `z-index:`
   Geben Sie die Eigenschaft `z-index` ein, gefolgt von einem Doppelpunkt.
4. `2;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.
5. Wiederholen Sie die Schritte 1 bis 4 für jedes Element, dem Sie einen z-index zuweisen möchten.

Listing 8.4 zeigt ein Beispiel, das Ergebnis sehen Sie in Abbildung 8.13. Beim natürlichen z-index wäre die Reihenfolge der Ebenen genau umgekehrt. Ebene1 hätte den z-index 0 und würde damit unter den anderen liegen. Ebene3 hätte den natürlichen z-index 2 und würde als oberste Ebene angezeigt werden. Durch das Festlegen des z-index wurde die Reihenfolge geändert.

```
<html><head>
<title>z-index</title>
<style type="text/css"><!--
div { width:100px; height:50px; border:2px
solid; padding:5px;
color:e00000; }
#hell { position:absolute; top:20px;
left:20px; background:#F0F0F0;
z-index:2; }
#mittel {position:absolute; top:50px;
left:50px;
z-index:1; background:#E0DCE0; }
#dunkel {position:absolute; top:80px;
left:80px;
z-index:0; background:#B0B0B0; }
--></style></head><body>
<div id="hell"><br>Ebene 1</div>
<div id="mittel"><br>Ebene 2</div>
<div id="dunkel"><br>Ebene 3</div>
</body></html>
```

Listing 8.4: z-index festlegen

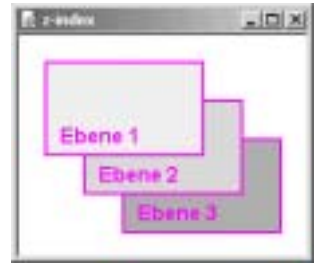

Abbildung 8.13: Durch das Festlegen des z-index wurde die Reihenfolge der Ebenen geändert.

Abbildung 8.14: Im natürlichen z-index liegt die Grafik über dem Text.

Abbildung 8.15: Die Grafik wurde mit `z-index:-1` versehen, alle anderen Elemente erhielten hier den `z-index:0`. Die Grafik liegt nun hinter dem Text.

Eingebettete Elemente

Sie können positionierte Elemente in anderen positionierten Elementen einbetten. In welcher Stapelreihenfolge die verschachtelten Elemente angezeigt werden, können Sie wieder durch den z-index festlegen.

Die Elternelemente haben im natürlichen z-index kleinere Nummern als die eingebetteten Elemente. Dadurch werden die Kindelemente über ihren Elternelementen angezeigt, verdecken diese ggf. also teilweise.

In Abbildung 8.14 sehen Sie einen Text, der den Inhalt des Elternelements bildet. Die Grafik ist in diesem Element eingebettet. Der z-index wurde nicht festgelegt, also verdeckt die Grafik einen Teil des Textes, da sie im natürlichen z-index eine höhere Nummer als das Elternelement hat.

Die Grafik soll aber als Hintergrund dienen, sie muss also unter dem Elternelement angezeigt werden. Dazu muss der z-index festgelegt werden. Für die Grafik wurde `z-index:-1` gesetzt, für alle anderen Elemente wurde hier der `z-index:0` verwendet. Solange sich die Elemente nicht überlappen, dürfen sie den gleichen z-index haben. Das Ergebnis sehen Sie in Abbildung 8.15.

Netscape 4 unterstützt die Stapelreihenfolge in verschachtelten Elementen nicht.

Absolute Elemente in relative Elementen einbetten

Wenn Sie ein absolutes Element in ein relatives Element einbetten, bezieht sich die absolute Position auf die linke obere Ecke des relativen Elternelements.

Wenn das Elternelement bewegt wird, behält das absolute Element seine Position im Elternelement bei. In Bezug auf den Seitenkörper ändert sich aber die Position.

Diese Art der Verschachtelung eignet sich immer dann, wenn Elemente zueinander stets die gleiche Position haben sollen, während sich die Position relativ zum Seitenkörper ändern darf oder sogar muss.

So können Sie z.B. Schatteneffekte für Schriftzüge realisieren, die sich mit dem Text verschieben, wenn Änderungen vorgenommen werden. Abbildung 8.16 zeigt einen solchen Schriftzug. In Abbildung 8.17 sehen Sie den Aufbau der einzelnen Schriftlagen. Der Text über dem Schriftzug ist länger; die Überschrift wird dadurch nach unten verschoben.

Abbildung 8.16: Schriftzug mit Schatteneffekt

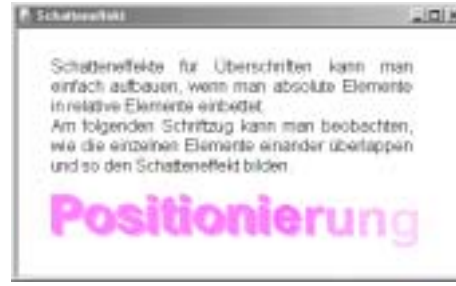

Abbildung 8.17: Hier sind die Schriftlagen deutlich zu erkennen. Durch das Einbetten in ein relatives Elternelement verschiebt sich der Schriftzug, wenn die Inhalte über dem Elternelement sich ändern.

```
body { margin:15px 20px; }

. headline { position:relative;
top:10px; left:0px; color:#000000;
font:bold 60px Arial,sans-serif;
z-index:6; }

.schriftzug { position:absolute;
top:0px; left:0px; color:#909090;
font:bold 50px Arial,sans-serif;
z-index:5; }

.konturrot { position:absolute;
top:1px; left:1px; color:#ff0000;
font:bold 50px Arial,sans-serif;
z-index:4; }

.schatten1 { position:absolute;
top:2px; left:3px; color:#909090;
font:bold 50px Arial,sans-serif;
z-index:3; }

.schatten2 { position:absolute;
top:3px; left:4px; color:#B0B0B0;
font:bold 50px Arial,sans-serif;
z-index:2; }

.schatten3 { position:absolute;
top:4px; left:5px; color:#d0d0d0;
font:bold 50px Arial,sans-serif;
z-index:1; }

.schatten4 { position:absolute;
top:5px; left:6px; color:#F0F0F0;
font:bold 50px Arial,sans-serif;
z-index:0; }

p { position:relative; margin:2px;
padding:2px; color:#000000; font:12pt
Arial,sans-serif;
text-align:justify;
z-index:100; }
```

Listing 8.5: Inhalt der Datei *headline.css*

So ist der Schatteneffekt entstanden:

Für den Schatteneffekt wurden in der externen Datei *headline.css* die Klassen definiert (siehe Listing 8.5).

Das Elternelement wird von der Klasse `.relElement` gebildet. Obwohl das Element in unserem Beispiel (siehe Abbildung 8.16) nur ein Leerzeichen und die absoluten Elemente enthält, wurde hier eine Schriftgröße angegeben, die mindestens so groß wie die eigentliche Überschrift sein sollte. Warum das so ist, wird später noch genau erklärt.

`.schriftzug` ist die Klasse für die Überschrift, die als oberste Lage angezeigt wird. Die Position wurde für links und rechts mit je null Pixel festgelegt. Diese Position bezieht sich auf die linke obere Ecke des relativen Elternelements.

`.konturrot` ist die Lage, die direkt unter dem Schriftzug angezeigt wird, sie ist nach rechts und unten um je 1 Pixel versetzt.

Die nachfolgenden Klassen stehen für die Schattenlagen. Jede ist zur darüber liegenden Lage ebenfalls nach rechts unten versetzt und in Grautönen abgestuft. `.schatten4` ist die unterste Schriftlage und hat den hellsten Grauton.

Für die unterste Schattenlage wird die Eigenschaft `z-index:0;` gesetzt, für die darüber liegenden Lagen wurde der z-index um jeweils eins erhöht.

Damit der Fließtext in jedem Fall als oberste Ebene angezeigt wird, erhält er die Eigenschaft `z-index:100`.

Der Quelltext der Webseite

Wie die Webseite aufgebaut ist, zeigt Listing 8.6. Das relativ positionierte Elternelement enthält die verschiedenen absoluten Elemente für den Schriftzug. Jedes dieser Elemente besteht im Beispiel aus dem Wort "Positionierung".

Wenn der Text im Absatz über dem relativen Elternelement kürzer oder länger wird, verschiebt sich der Schriftzug entsprechend nach oben oder unten, das haben Sie bereits in Abbildung 8.16 und Abbildung 8.17 gesehen.

```
<html><head>
<title>Schatteneffekt</title>
<link rel="stylesheet" href="headline.css"
type="text/css">
</head>
<body>
<p>Schatteneffekte für Überschriften kann
man einfach aufbauen, wenn man absolute
Elemente in relative Elemente einbettet.
</p>

<div class="headline">
 
<div class="schriftzug">
Positionierung
</div>
<div class="konturrot">
Positionierung
</div>
<div class="schatten1">
Positionierung
</div>
<div class="schatten2">
Positionierung
</div>
<div class="schatten3">
Positionierung
</div>
<div class="schatten4">
Positionierung
</div>

</div>

<p>Nachfolgender Text</p>
</body></html>
```

Listing 8.6: Einbetten des Schriftzugs in ein relatives Elternelement

Abbildung 8.18: Hier fehlen Leerzeichen und Schriftgröße im relativen Elternelement. Die Überschrift wird in einer helleren Farbe angezeigt, damit der darüber liegende Text besser zu sehen ist.

Abbildung 8.19: Das Leerzeichen wurde gesetzt, aber die Schriftgröße im Elternelement fehlt. Der Text wird immer noch auf der Überschrift angezeigt.

Abbildung 8.20: Nur wenn das Leerzeichen und eine ausreichende Schriftgröße gesetzt sind, wird der nachfolgende Text an der richtigen Position angezeigt.

Nachfolgende relative Elemente

Wie sieht die Sache aus, wenn nach oder in dem relativen Elternelement weitere relative Elemente, z.B. Textabsätze, stehen?

Es wurde bereits erwähnt, dass für das Elternelement eine Schriftgröße gewählt wurde, die mindestens der Größe des Schriftzugs entspricht. Außerdem muss das Elternelement ein Leerzeichen oder einen Zeilenumbruch enthalten. Beide zusammen dienen dazu, nachfolgende relative Elemente mit dem richtigen Abstand anzuzeigen.

Wenn Sie das Leerzeichen bzw. den Umbruch weglassen, wird ein nachfolgendes relatives Element in jedem Fall auf dem Schriftzug angezeigt, egal ob die Schriftgröße im Elternelement gesetzt ist oder nicht (siehe Abbildung 8.18). Das Leerzeichen allein schafft auch keine Abhilfe (siehe Abbildung 8.19).

Erst wenn Leerzeichen und eine Schriftgröße gesetzt sind, wird der nachfolgende Text richtig angezeigt. Je größer die definierte Schriftart im Elternelement, desto größer der Abstand zwischen dem absoluten Schriftzug und dem nachfolgenden relativen Element. Im Beispiel hat der Schriftzug eine Höhe von 50px, für das Elternelement wurde die Schriftgröße 60px gewählt.

Alternativ können Sie auch eine transparente Grafik, die auf die erforderliche Höhe gedehnt wird, als Abstandhalter verwenden.

Relative Elemente in absolute Elemente einbetten

Sie können relative Elemente auch in absolute Elemente einbetten. Damit haben Sie die Möglichkeit, Bereiche pixelgenau zu positionieren und innerhalb dieser Bereiche mit relativen Positionierungen Effekte zu erzielen.

In Abbildung 8.21 wurde ein zweispaltiges Layout verwendet. Die beiden Spalten sind absolut positioniert. Innerhalb der Spalten wurde mit relativ positionierten Bereichen gearbeitet.

Die Seite in Abbildung 8.22 hat die gleichen Inhalte, hier wurden aber die Breiten der beiden Elemente vergrößert und die Position des zweiten Elements verändert.

Abbildung 8.21: Ansicht des zweispaltigen Layouts in Netscape 4

Abbildung 8.22: Hier wurden die Positionen und die Breite der Elemente geändert.

```
body, p { font:12pt Arial; text-align:
justify; margin:2px; padding:3px; }

.spalteeins { position:absolute;
left:10px; top:10px; width:200px;
padding:10px; }

.spaltezwei { position:absolute;
left:230px; top:10px; width:200px;
padding:10px; }

.inhalt { position:relative; top:5px;
width:60px; float:left; font:9pt Verdana;
padding:10px; }
```

Listing 8.7: Inhalt der Datei *layout.css* für das Layout in Abbildung 8.21.

```
body, p { font:12pt Arial; text-align:
justify; margin:2px; padding:3px; }

.spalteeins { position:absolute;
left:20px; top:10px; width:300px;
padding:10px; }

.spaltezwei { position:absolute;
left:100px; top:160px; width:300px;
padding:10px; }

.inhalt { position:relative; top:5px;
width:60px; float:left; font:9pt Verdana;
padding:10px; }
```

Listing 8.8: Für das Layout in Abbildung 8.22 wurden die Breiten der Elemente geändert und das zweite Element nach unten rechts versetzt. Die Datei wurde als *layout2.css* gespeichert.

So ist das Layout entstanden:

Für das Layout in Abbildung 8.21 wurden in der Datei *layout.css* die Klassen definiert (siehe Listing 8.7).

Die beiden absolut positionierten Elternelemente werden von den Klassen `.spalteeins` und `.spaltezwei` gebildet. Die Breite wurde auf 200px festgelegt, die `top`-Position ist in diesem Fall für beide Elemente gleich. `.spaltezwei` wurde mit `left:230px;` soweit nach rechts verschoben, dass zwischen den beiden Spalten ein Leerraum bleibt.

Für die Grafiken der Initialen und deren Beschriftung wurde die Klasse `.inhalt` angelegt. Wenn die Inhalte der Spalten verschiedene Formatierungen haben sollen, muss für jede Spalte eine Inhaltsklasse angelegt werden. Durch `float:left;` fließt Text, der nach dem Element folgt, um die Initialen herum.

Für das Layout in Abbildung 8.22 wurde die Breite der beiden Elternelemente geändert. `spaltezwei` wurde weiter unten rechts positioniert (siehe Listing 8.8). Andere Änderungen sind nicht erforderlich.

Der Quelltext der Webseiten

Für die Webseiten in Abbildung 8.21 und Abbildung 8.22 wird der gleiche Quelltext verwendet. Die veränderten Layouts kommen nur durch die Änderungen in der externen CSS-Datei *layout.css* zustande.

Den Quelltext sehen Sie in Listing 8.9. Hier wurde die Datei *layout2.css* referenziert; das Ergebnis zeigt Abbildung 8.22. Für das Beispiel in Abbildung 8.21 wurde an dieser Stelle die Datei *layout.css* eingebunden.

```
<html><head>
<title>Position</title>
<link rel="stylesheet" href="layout2.css"
type="text/css">
</head>
<body>

<div class="spalteeins">

<div class="inhalt">
<img src="b.gif" width="35" height="32"
border="0" alt="B">
<br>Initial </div>

<p>Hier steht Text, der rechts vom Initial
und der Beschriftung angezeigt wird.</p>
<p>Der nächste Absatz mit Fließtext.
Weiterer Text, und noch mehr. </p>
</div>

<div class="spaltezwei">

<div class="inhalt">
<img src="d.gif" width="35" height="32"
border="0" alt="D">
<br>Initial
</div>
<p>Hier steht Text, der rechts vom Initial
und der Beschriftung angezeigt wird.
</p><p>Der nächste Absatz mit Fließtext.
Weiterer Text, und noch mehr. </p>
</div>

</body></html>
```

Listing 8.9: Der Quelltext für beide Beispiele

9

ANZEIGE VON ELEMENTEN

Mit CSS können Sie die Anzeige der Elemente beeinflussen. Man kann bestimmen, wie ein Element angezeigt wird, ob es sichtbar oder unsichtbar ist oder ob z.B. nur Ausschnitte einer Grafik angezeigt werden.

Einige Eigenschaften, die die Anzeige von Elementen beeinflussen, werden erst dann wichtig, wenn Sie DHTML einsetzen. Dabei wird die Sichtbarkeit der Elemente mit JavaScript beeinflusst.

In DHTML ist die Anzeige von Elementen einer der Dreh- und Angelpunkte, dort werden diese CSS-Eigenschaften ständig gebraucht.

display vs. visibility

Es gibt zwei verschiedene Eigenschaften, die die Anzeige von Elementen beeinflussen: `display` und `visibility`.

Display bestimmt die *Anzeigeart* eines Elements. Mit dieser Eigenschaft können Sie z.B. erzwingen, dass ein Inline-Tag als Blocklevel-Tag angezeigt wird. Einige Werte von `display` werden von den gängigen Browser nicht oder nur unvollständig unterstützt. Mit `display:none` können Sie ein Element vollständig von der Webseite verschwinden lassen. Das Element steht zwar im Quelltext, wird aber nicht angezeigt. Ihm wird auch kein Raum auf der Webseite gegeben. Es deutet also nichts darauf hin, dass das Element überhaupt vorhanden ist (siehe Abbildung 9.2).

`visibility` ist nur für die *Sichtbarkeit* eines Elements zuständig. Mit `visibilty:hidden` können Sie ein Element ebenfalls unsichtbar machen. Im Gegensatz zu `display:none` wird für das Element aber entsprechend viel Platz auf der Seite gelassen. Es wird ein rahmenloser Platzhalter gesetzt (siehe Abbildung 9.3).

Mit welcher der beiden Eigenschaften ein Element unsichtbar gemacht wird, hängt davon ab, was man erreichen will. `visibility` wird in der Praxis häufiger eingesetzt als `display`. Das liegt u.a. daran, dass die Werte von `visibility` in den gängigen Browsern mehr Unterstützung finden als die `display`-Werte.

Abbildung 9.1: So sieht die Seite aus, wenn der Schriftzug sichtbar ist.

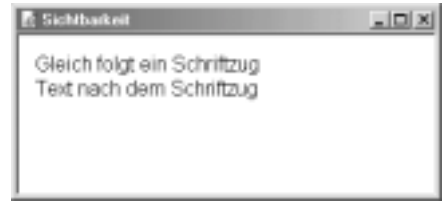

Abbildung 9.2: Mit `display:none` ist der Schriftzug aus der Webseite verschwunden.

Abbildung 9.3: Mit `visibility:hidden` ist der Schriftzug unsichtbar geworden, es wird aber entsprechend viel Platz gelassen.

Abbildung 9.4: Opera zeigt alle Werte an.

Abbildung 9.5: N4 zeigt nur `none` richtig an.

Abbildung 9.6: IE5 unterstützt `list-item` nicht.

Anzeigeart eines Elements

Mit der Anzeigeart für ein Tag können Sie bestimmen, wie das Element vom Browser tatsächlich angezeigt wird. Die Eigenschaft heißt `display`, erlaubte Werte sind `block`, `inline`, `list-item` und `none`.

- `block` erzwingt die Anzeige als Blocklevel-Tag, auch wenn es sich um ein Inline-Tag handelt. Vor und nach dem Element werden Umbrüche angezeigt.
- `inline` erzwingt die Anzeige als Inline-Tag und unterdrückt die Anzeige der Umbrüche bei Blocklevel-Tags.
- `list-item` zeigt das Element wie ein Listenelement an. Die Anzeige entspricht dem Wert `block`, zusätzlich wird ein Bullet gesetzt.
- `none` verbirgt das Element völlig. `none` ist der einzige Wert, der von allen Browsern richtig angezeigt wird.

Opera 5 zeigt alle Werte richtig an (siehe Abbildung 9.4).

Netscape4 unterstützt außer `none` keinen der Werte. `block` und `list-item` werden ansatzweise definiert, aber nicht richtig (siehe Abbildung 9.5).

IE5 bietet keine Unterstützung für `list-item` (siehe Abbildung 9.6).

So bestimmen Sie die Anzeigeart eines Elements:

1. `display:`
   Geben Sie die Eigenschaft `display` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 9.1 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 9.7.

✓ Tipp:

- Beim Einsatz von `none` sollten Sie vorsichtig sein. Wenn der Wert auf ein Element angewendet wird, in das andere Elemente eingebettet sind, werden auch die Kindelemente unsichtbar.

```
<html><head><title>Display</title>
<style type="text/css">
<!--
p,body { font:12pt Arial,sans-serif;
margin:5px; }
.eins { display:block; }
.zwei { display:inline; }
.drei { display:list-item; }
-->
</style>
</head><body>
<p>Ein <span class="eins">Inline-Tag
</span> als Block </p>

Jetzt folgt das Tag h3.
<h3 class="zwei">h3 ist ein Blocklevel-
Tag, </h3> das hier als Inline-Tag ange-
zeigt wird.

<p>Ein <span class="drei">Inline-Tag
</span> als Listenelement </p>.
</body></html>
```

Listing 9.1: Anzeigeart eines Elements

Abbildung 9.7: Ansicht im Browser

Abbildung 9.8: Mit dem Wert `inherit` wird die Sichtbarkeit seines Elternelements übernommen.

Sichtbarkeit von Elementen

Mit der Eigenschaft `visibility` können Sie bestimmen, ob ein Element sichtbar ist oder nicht. Erlaubte Werte sind `visible`, `hidden` und `inherit`.

- `visible` zeigt das Element an.
- `hidden` macht das Element unsichtbar. Auf der Webseite wird aber ein Leerraum angezeigt, der dem Platzbedarf des Elements entspricht.
- `inherit` ist der englische Begriff für *vererbt*. Bei diesem Wert übernimmt das Element den Wert seines Elternelements.

In Abbildung 9.8 sehen Sie, wie sich der Wert `inherit` auswirkt.

In Netscape 4 kennt noch zwei spezifische Werte, die aber nicht zum Standard gehören: `hide` und `show`. Sie entsprechen den Werten `hidden` und `visible`. N4 unterstützt die Standardwerte, wenn die Positionierungsart gesetzt ist, übersetzt sie aber in seine spezifischen Werte. Probleme können dabei entstehen, wenn mit JavaScript der Sichtbarkeitsstatus eines Elements festgestellt werden soll (mehr dazu in Kapitel 13).

So bestimmen Sie die Sichtbarkeit eines Elements:

1. `visibility:`
   Geben Sie die Eigenschaft `visibility` ein, gefolgt von einem Doppelpunkt.
2. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 9.2 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 9.9.

✓ Tipps:

- Setzen Sie immer auch die Eigenschaft `position` für die Positionierungsart. Netscape 4 zeigt die Werte für die Sichtbarkeit sonst nicht an.
- Wenn Sie die Sichtbarkeit des Elements mit JavaScript beeinflussen wollen, sollten Sie statt Klassen immer IDs verwenden.

```
<html><head>
<title>Visibility</title>
<style type="text/css">
<!--
p,body { font:12pt Arial,sans-serif;
margin:5px; }
h2 { font:bold 16pt Georgia,serif;
margin:5px; }
#eins { position:relative;
visibility:visible; }
#zwei { position:relative;
visibility:hidden; }
-->
</style>
</head><body>
<h2>Sichtbar oder unsichtbar?</h2>
<p id="eins">Dieser Absatz ist sichtbar.
</p>
<p id="zwei">Dieser Absatz ist
unsichtbar.</p>
<p id="eins">Der Absatz über diesem ist
unsichtbar.</p>
</body></html>
```

Listing 9.2: Sichtbarkeit eines Elements

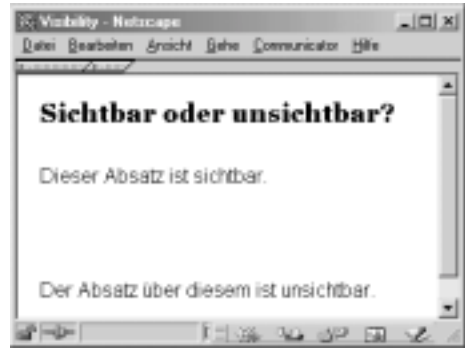

Abbildung 9.9: Unsichtbarer Absatz im Browser

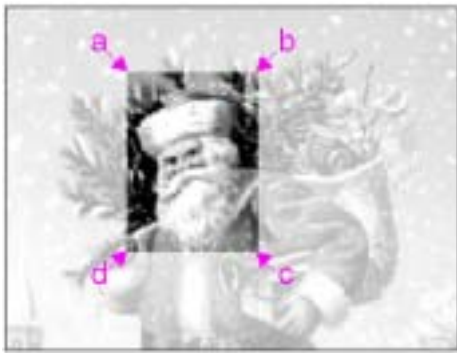

Abbildung 9.10: Hier sind die Punkte bezeichnet. Die Punkte a und c werden von der oberen Kante aus gemessen, b und d von der linken Elementkante aus.

✓ Tipps:

- Opera unterstützt Clipping gar nicht.
- Netscape 4 und Internet Explorer 5 unterstützen Clipping nur, wenn das Element absolut positioniert ist, aber nicht bei der relativen Positionierung.
- Netscape hat massive Schwierigkeiten, Clipping in manchen Tags anzuwenden. Dazu gehört z.B. auch das `<img>`-Tag. Umschließen Sie Elemente, die geclippt werden sollen, wie `<span>` oder `<div>` und wenden Sie das Clipping auf diese Tags an.
- Das Element beansprucht nach dem Clipping ebenso viel Platz wie normal. Der geclippte Bereich wird also nicht nach oben oder links gerückt, sondern bleibt an seinem Platz.
- Wenn das Element von einem Rahmen umschlossen wird, verschwindet dieser beim Clipping.

Sichtbarer Bereich eines Elements

Mit CSS können Sie festlegen, welcher Bereich eines Elements angezeigt werden soll. Die Eigenschaft zum Festlegen des sichtbaren Bereichs heißt `clip`. Im Fachjargon hat sich der Begriff Clipping eingebürgert, wenn ein sichtbarer Bereich festgelegt wird.

Erlaubte Werte sind `rect(a b c d)` und `auto`. Beim Wert `auto` wird das Element an der entsprechenden Seite bis zur Elementgrenze angezeigt.

Der Wert `rect` besteht aus dem Schlüsselwort und vier Pixelwerten, die direkt nach `rect` in runden Klammern stehen. Vergleichen Sie dazu auch Abbildung 9.10; dort sind die Punkte entsprechend bezeichnet:

- Der erste Pixelwert `a` bezeichnet den linken oberen Eckpunkt, gemessen an der oberen Elementgrenze.
- Der zweite Pixelwert `b` bezeichnet den rechten oberen Eckpunkt, gemessen an der linken Elementgrenze.
- Der dritte Pixelwert `c` bezeichnet den rechten unteren Eckpunkt, gemessen an der oberen Elementgrenze.
- Der vierte Pixelwert `d` bezeichnet den linken unteren Eckpunkt, gemessen an der linken Elementgrenze.

Der komplette Wert sieht also z.B. so aus:

```
rect(40px 155px 150px 75px)
```

Da hier nur Pixelwerte erlaubt sind, können Sie die Einheit `px` auch weglassen.

So legen Sie einen sichtbaren Bereich für ein Element fest:

1. `position:absolute; top:10px;`
   Notieren Sie die absolute Positionierung und mindestens einen Startwert.
2. `clip:`
   Geben Sie die Eigenschaft `clip` ein, gefolgt von einem Doppelpunkt.
3. `rect(`
   Schreiben Sie dahinter das Schlüsselwort `rect`, gefolgt von der öffnenden runden Klammer.
4. `40px 155px 150px 75px);`
   Setzen Sie die gewünschten Werte für die Eckpunkte des Rechtecks. Nach dem letzten Wert schreiben Sie die schließende runde Klammer, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 9.3 zeigt ein Beispiel, das Ergebnis im Browser sehen Sie in Abbildung 9.11.

```
<html><head>
<title>Clipping</title>
<style type="text/css">
<!--
.ausschnitt { position:absolute; top:10px;
left:10px;
clip:rect(40px 155px 150px 75px); }
.text { position:absolute; top:200px;
left:10px; }
p { position:relative; font:12pt
Arial,sans-serif; }
-->
</style></head>
<body>
<div class="ausschnitt">
<img src="motiv.jpg" width="275"
height="210" border="0">
<p>Nun folgt Text</p>
</div>
<div class="text"><p>Nun folgt Text</p>
</div>
</body></html>
```

Listing 9.3: Sichtbarer Bereich mit clipping

Abbildung 9.11: Eine geclippte Grafik

Abbildung 9.12: Bei den Elementgrößen definiert IE5 die Werte richtig. Von oben nach unten: `hidden`, `scroll`, `visible`.

Was geschieht mit Überhängen?

Beim Clipping entstehen normalerweise immer Überhänge. Damit sind Inhalte gemeint, die über den festgelegten Bereich hinausragen. Überhänge können auch entstehen, wenn Sie die Breite und Höhe eines Elements festlegen und die Inhalte dann größer sind als dieser Bereich.

Mit der Eigenschaft `overflow` können Sie bestimmen, wie diese Überhänge angezeigt werden. Dies wurde bereits in Kapitel 7 beim Festlegen der Elementhöhe kurz erwähnt. Bei der Höhe eines Elements spielt `overflow` eine wichtige Rolle, da fast alle Browser `height` ignorieren, wenn `overflow` nicht gesetzt ist.

Erlaubte Werte für `overflow` sind `auto`, `visible`, `hidden` und `scroll`.

- `auto` überlässt dem Browser die Entscheidung, was mit dem Überhang passieren soll.
- `visible` erzwingt die Anzeige des Überhangs. Das Element wird soweit vergrößert, dass die Inhalte angezeigt werden. Dabei werden Größenwerte außer Kraft gesetzt, Clipping wird dann ebenfalls ignoriert.
- `hidden` schneidet die Überhänge ab, es erscheinen keine Scrollbalken.
- `scroll` verbirgt die Überhänge, es werden Scrollbalken angezeigt, über die der Besucher die überhängenden Inhalte anzeigen kann.

`overflow` kann nur dann verwendet werden, wenn die Positionierungsart (`absolute` oder `relative`) in der gleichen Stilregel festgelegt ist.

So kontrollieren Sie Überhänge:

1. `position:relative; top:10px;`
   Notieren Sie die Positionierungsart und mindestens einen Startwert.
2. `width:150px; height:100px;`
   Notieren Sie die Elementgrößen.
3. `overflow:`
   Geben Sie die Eigenschaft `overflow` ein, gefolgt von einem Doppelpunkt.
4. `Wert;`
   Schreiben Sie dahinter den gewünschten Wert, gefolgt von einem Semikolon, das die Stilregel abschließt.

Listing 9.4 zeigt ein Beispiel, das Ergebnis im Browser haben Sie bereits in Abbildung 9.12 gesehen.

✓ Tipps:

- `overflow` gilt auch für das Clipping, wird dort von den Browsern aber nicht unterstützt.
- Netscape 4 unterstützt nur `overflow:visible` im Zusammenhang mit der Elementgröße. `hidden` und `scroll` werden nicht definiert.
- Opera kennt den Wert `scroll` bei den Elementgrößen nicht.
- IE definiert bei den Größen die Werte richtig.

```
<html><head>
<title>Overflow</title>
<style type="text/css">
<!--
body { margin:0px; background:#FFFFFF; }
.eins { position:relative; top:10px;
left:10px; width:150px; height:100px;
overflow:hidden; }
.zwei { position:relative; top:10px;
left:10px; width:150px; height:100px;
overflow:scroll; }
.drei { position:relative; top:10px;
left:10px; width:150px; height:100px;
overflow:visible; }
div { margin:10px; }
-->
</style></head><body>
<div class="eins">
<img src="motiv2.jpg" width="275"
height="210" border="0">
</div>
<div class="zwei">
<img src="motiv2.jpg" width="275"
height="210" border="0">
</div>
<div class="drei">
<img src="motiv2.jpg" width="275"
height="210" border="0">
</div>
</body></html>
```

Listing 9.4: Überhänge kontrollieren

10

DHTML-EINFÜHRUNG

Sie wissen nun, welche Möglichkeiten Sie mit CSS zum Formatieren Ihrer Seiten haben. Zusammen mit JavaScript bilden CSS die Grundlage für DHTML (Dynamisches HTML).

DHTML ist ein breit gefächertes Gebiet, und man könnte allein zu diesem Thema ein Buch mit einem Seitenumfang von tausend Seiten und mehr schreiben.

So viel Platz steht in diesem Buch nicht zur Verfügung, aber Sie lernen die erforderlichen Grundlagen kennen, um pfiffige Effekte für Ihre Webseiten zu realisieren.

Wenn Sie die Grundlagen verstanden haben, können Sie sich in das DHTML-Vergnügen stürzen. Es gibt unzählige Seiten, auf denen fertige Scripts zur Verfügung stehen, die Sie mit dem Gelernten nach Ihren eigenen Vorstellungen leicht verändern können. Mit etwas Übung schreiben Sie dann schon bald Ihre eigenen Scripte.

Was ist DHTML?

Da DHTML für *Dynamic HTML* steht, sollte man eigentlich annehmen, dass es sich hier wie bei HTML oder JavaScript um einen Sprachstandard handelt.

Nun, dem ist nicht so. Es gibt keinen Standard, der festlegt, was DHTML ist. DHTML ist keine HTML-Erweiterung wie CSS und auch keine eigene Sprache. Es gibt keine offizielle Definition zu DHTML. Dennoch hat sich unter den Nutzern eine inoffizielle Definition durchgesetzt, die zumindest einige Anhaltspunkte bietet (siehe hierzu den Infokasten *Die inoffizielle Definition)*.

Eigentlich ist DHTML ein Kunstbegriff, den sich die Marketingstrategen der Browserhersteller ausgedacht haben, um bestimmte Browsertechnologien an den Mann (und die Frau) zu bringen. Diese verschiedenen Browsertechnologien sind es aber auch, die uns die Suppe versalzen.

Die inoffizielle Definition

DHTML ermöglicht es, Elemente der Webseite während der Anzeige dynamisch zu beeinflussen oder zu ändern. Das kann automatisch geschehen oder durch Interaktion eines Besuchers.

Wie HTML und CSS soll DHTML plattform- und browserübergreifend arbeiten. (Es ist bekannt, wie schwierig das schon mit HTML und CSS ist. In der Praxis wird sich diese Anforderung also schwer erfüllen lassen.)

Für DHTML sind keine Browser-Plug-Ins erforderlich.

DHTML nutzt HTML, CSS und Scriptsprachen wie JavaScript, um dynamische Effekte auf der Webseite zu erzeugen.

Für die dynamischen Effekte und Interaktionen muss nicht ständig auf den Server zugegriffen werden. DHTML ist clientseitig.

Abbildung 10.1: Die Netscape-Seite zu DHTML

Abbildung 10.2: Die Microsoft-Seite zu DHTML

Der Kampf der Giganten

Netscape und Internet Explorer sind immer noch die beiden Marktführer unter den Browsern. Die Hersteller haben von Anfang an jeder sein eigenes Süppchen gekocht und in die Browser Technologien integriert, die andere Browser nicht unterstützen.

Mit Erscheinen der 4er-Versionen von Netscape und Internet Explorer ist dann ein regelrechter Krieg ausgebrochen, der einem Webdesigner das Leben erschwert.

Mittlerweile hat das W3C als schlichtende Partei in diesen Krieg eingegriffen und das Document Object Model (DOM) entwickelt. In neueren Browserversionen wird das DOM des W3C nach und nach integriert. Es wird zwar noch einige Jahre dauern, bis man die Altlasten los geworden ist und die Browser tatsächlich so arbeiten, dass man eine einheitliche Basis hat, aber immerhin: Es ist Land in Sicht.

Doch inzwischen müssen wir uns leider mit den Gegebenheiten abfinden und die Eigenheiten der Browser in Kauf nehmen.

CrossBrowser DHTML

Die Frage "Was ist CrossBrowser DHTML?" zu beantworten, ist leichter, als eine Antwort auf die allgemeine Frage "Was ist DHTML?" zu finden.

Während Microsoft im IE dynamische Filter, Datenanbindung und Scriptsprachen wie JScript und VBScript intergrierte, wartet Netscape mit JSS (JavaScript Style Sheets) und Layern auf.

Doch zum Glück gibt es Bereiche, in denen sich die dynamischen Technologien überlappen (siehe Abbildung 10.3).

Die drei Standards, die von jedem Browser mehr oder weniger interpretiert werden, sind CSS, JavaScript und DOM. Sie bilden zusammen mit HTML die Grundlage für browserübergreifendes DHMTL. Im Fachjargon hat sich der englische Begriff *CrossBrowser DHTML* durchgesetzt.

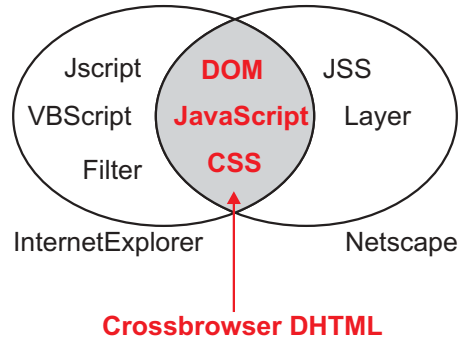

Abbildung 10.3: Überlappung der dynamischen Technologien

✓ Tipps:

- HTML und CSS kennen Sie bereits. Fundierte Kenntnisse in JavaScript sind erforderlich, wobei es auch einige DHTML-Effekte gibt, für die Sie nur wenig JavaScript brauchen.
- Dieses Buch ist keine JavaScript-Referenz. Es werden zwar einige Kenntnisse vermittelt, aber wenn Sie noch nie Kontakt mit JavaScript hatten, empfehle ich Ihnen, sich zuerst die Grundkenntnisse anzueignen. Einen leichten Einstieg in das Thema JavaScript bietet Ihnen das Buch *JavaScript*, ISBN 3-8272-5960-6 aus der Buchreihe *Digital Studio One*.

Die Konkurrenz: Flash

Es gibt andere dynamische Technologien, die mit DHTML konkurrieren – allen voran Macromedias Flash.

Beim Einbinden von Multimedia-Inhalten hat Flash sicher die Nase vorn, da es dort mehr Möglichkeiten gibt.

Andererseits ist für Flash teure Software erforderlich, die nicht einfach zu handhaben ist. Flash-Dateien erreichen mitunter Größen, die die Downloadzeiten enorm verlängern, wenn man nicht gerade eine DSL-Anbindung ins Internet hat. Hinzu kommt, dass Flash Plug-Ins für den Browser erforderlich macht.

Flash hat wie DHTML Stärken und Schwächen . Welche der beiden Technologien verwendet wird, bleibt jedem selbst überlassen.

Stärken und Schwächen

Wie jede Technologie hat DHTML Stärken und Schwächen, wobei die Vorteile bei weitem überwiegen.

Vorteile:

1. *Breite Unterstützung:* Die meisten Surfer verwenden Internet Explorer 5 und Netscape 4 oder neuere Versionen. Aber auch Browser wie Opera oder iCab unterstützen DHTML bzw. es wird daran gearbeitet.
2. *Keine Plug-Ins erforderlich:* Für DHTML braucht man keine speziellen Browser-Plug-Ins. Browser, die CSS, JavaScript und das DOM unterstützen, zeigen DHTML an.
3. *Schnelle Entwicklung und Verbreitung:* Es gibt Webseiten, die sich darauf spezialisiert haben, neue DHTML-Effekte zu entwickeln und zu verbreiten. Außerdem gibt es viele Foren, in denen man sich bei Bedarf Hilfe holen kann.
4. *Kurze Einarbeitungszeiten:* Wenn Sie HTML, CSS und JavaScript kennen, ist es nur noch ein kleiner Schritt, bis Sie auch DHTML beherrschen.
5. *Kleine Dateigrößen:* DHTML besteht aus ASCII-Text, der die Dateigrößen gering hält. Die Anzeige im Browser erfolgt dadurch generell schneller als bei Flash oder Java-Applets.
6. *Schnelle Performance:* Durch die kleineren Dateigrößen ergeben sich gleichzeitig auch kürzere Downloadzeiten. Elemente können ein- und ausgeblendet werden, ohne dass neue Dateien vom Server geholt werden müssen.

Nachteile

- *Strikte Syntaxregeln:* HTML geht großzügig mit fehlenden oder falschen Tags um. CSS und JavaScript sind hingegen sehr kleinlich bei der Syntax. Ein Befehlszeichen zu viel oder zu wenig kann dazu führen, dass das ganze Script nicht mehr funktioniert.
- *Browserschwächen:* Je nach Browser werden JavaScript und CSS unterschiedlich unterstützt. Oft gibt es sogar Unterschiede beim gleichen Browser auf verschiedenen Plattformen. Hinzu kommen unterschiedliche DOM-Technologien. Selbst bei CrossBrowser DHTML wird es immer wieder Browser geben, die manche Dinge nicht beherrschen.
- *Browserbugs:* Viele Browser haben in Bezug auf CSS und JavaScripts einige Bugs. Berühmt-berüchtigt ist z.B. der Resize-Bug in Netscape. Wenn Sie externe CSS oder JavaScript verwenden, zeigt Netscape diese nach der Größenänderung des Browserfensters nicht mehr an. Lädt man die Seite neu, funktioniert alles wieder normal.

11

DOCUMENT OBJECT MODEL

Das Document Object Model (DOM) ermöglicht es, auf Objekte (Elemente mit einer eindeutigen ID) zuzugreifen und sie dann mit Hilfe von JavaScript zu ändern und zu bewegen.

Neuere Browserversionen wie Netscape 6 und Internet Explorer 5 verwenden das DOM, das vor einiger Zeit vom W3C entwickelt wurde und stellen so eine gemeinsame Basis bereit. Auch die Opera-Versionen 5 und 6 verwenden bereits das DOM des W3C.

Netscape 4 und Internet Explorer 4 verwenden leider unterschiedliche DOM-Technologien. Man kann sie aber durchaus auf einen gemeinsamen Nenner bringen. Wie man das macht, lernen Sie in diesem Kapitel.

Was ist das DOM?

Die Bezeichnung *Document Object Model* hört sich ziemlich abstrakt an und hat leider schon viele Webdesigner abgeschreckt. Dazu besteht aber kein Grund, denn das DOM ist eigentlich ganz einfach zu verstehen.

Stellen Sie sich vor, Sie wollen eine Adress-Datenbank aufbauen, deren Grundlage eine Textdatei ist. In dieser Textdatei trennt ein Punkt die einzelnen Felder voneinander. Die Daten werden in absteigender Reihenfolge angegeben, also Land, Ort usw. bis zur letzten Angabe, dem Namen des Empfängers. Die Textzeile für eine Adresse würde dann so aussehen:

```
land.ort.plz.strasse.hausnummer.name
```

Wenn Sie ein Objekt auf einer Webseite ansprechen möchten, müssen Sie ebenfalls eine Adresse angeben. Sie ist genauso aufgebaut wie die Adressangabe in der Textdatei für die Datenbank. Der Unterschied liegt darin, dass Sie hier keine Stadt, Straße usw. angeben müssen, sondern die Bezeichnungen, über die ein Browser ein Objekt finden kann. Beispiel:

```
window.document.images.name
```

Die Bezeichnungen beruhen auf einer Objekt-Hierarchie, die Ihnen aus JavaScript bekannt sein sollte (siehe Abbildung 11.1).

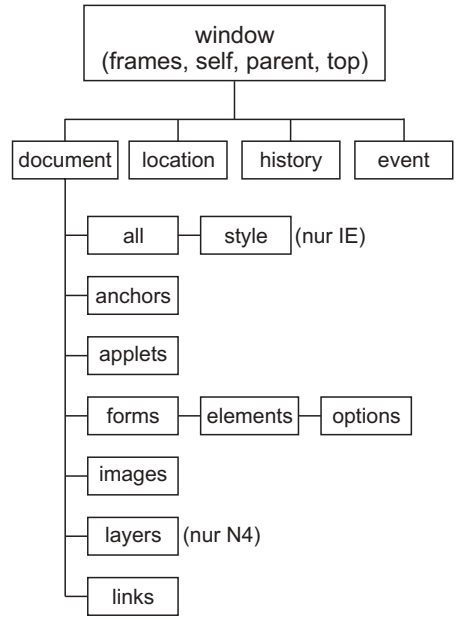

Abbildung 11.1: Die Objekt-Hierarchie

✓ Tipp:

- Das oberste Objekt der Hierarchie ist `window`. Wenn Elemente im gleichen Fenster angesprochen werden, können Sie diese oberste Ebene weglassen. Aus der DOM-Adresse `window.document.images.bild1` wird dann z.B. `document.images.bild1`.

Was mit DOM möglich ist	Browser
3D-Position eines Elements ändern (z-index)	IE4, N4
Elemente verbergen oder anzeigen (visibility)	IE4, N4
Die Position eines Objekts ändern (position)	IE4, N4
Objekte automatisch bewegen	IE4, N4
Objekte durch Interaktion des Besuchers bewegen	IE4, N4
Ändern des Anzeigebereichs (Clipping)	IE5, N4
CSS-Eigenschaften eines Elements ändern, während das Objekt bereits auf dem Bildschirm angezeigt wird.	IE4, N6

Tabelle 11.1: In der Tabelle sind die Browserversionen angegeben, die als erste die Möglichkeiten unterstützen.

Abbildung 11.2: Die Seite des W3C zum DOM

Eindeutige Objekt-Bezeichnungen durch Namen und IDs

Das DOM gibt also die Adresse vor, über die Sie auf Objekte zugreifen können. Dazu braucht jedes Element, das angesprochen werden soll, eine eindeutige Bezeichnung. Sobald ein Element bezeichnet wurde, ist es ein Objekt.

Die Bezeichnung ist eine ID oder ein Name. Beide werden im entsprechenden HTML-Tag als Attribut angegeben. Doch wann setzt man Namen ein, wann IDs?

IDs oder Namen?

Grundsätzlich müssen alle Objekte eine ID haben. Einzige Ausnahme sind Tags, die laut HTML-Spezifikation das Attribut `name` führen können, z.B. Formularfelder oder das `<img>`-Tag. Theoretisch reicht auch bei diesen Tags die ID aus, Internet Explorer erkennt aber z.B. nur Grafiken, die Namen haben. In diesen Tags kann man beides zusammen angeben, Name und ID.

Welche Möglichkeiten bietet das DOM?

Mit der eindeutigen Objekt-Adresse kann man auf ein Objekt Einfluss nehmen. Welche Möglichkeiten das DOM dabei bietet, sehen Sie in Tabelle 11.1.

Verschiedene DOMs

Bisher haben wir vom DOM geredet, das vom W3C entwickelt wurde (siehe Abbildung 11.2). Es wird in neueren Browsern integriert, um endlich eine einheitliche Basis zu schaffen.

Außerdem gibt es noch die browserspezifischen DOM-Technologien von Netscape 4 und Internet Explorer 4. Diese beiden DOMs werden wir uns später in diesem Kapitel noch genauer ansehen.

Objekte erzeugen

Wenn ein Element durch eine ID eine eindeutige Bezeichnung erhalten hat, ist es ein Objekt, auf das über das DOM zugegriffen werden kann.

So erzeugen Sie ein Objekt:

1. `#objekt { position:relative; }`
   Schreiben Sie eine ID-Stilregel (siehe Kapitel 2), in der die Positionierungsart angegeben ist. Wenn das Objekt bestimmte CSS-Stile haben soll, fügen Sie die gewünschten Stildefinitionen der Stilregel hinzu.
2. `<div id="objekt"> … </div>`
   Wenden Sie die ID in einem HTML-Tag an. Vorzugsweise sollten Sie `<div>`-Tags für absolute Elemente und `<span>`-Tags für relative Elemente verwenden.

Listing 11.1 zeigt ein Beispiel, das Ergebnis sehen Sie in Abbildung 11.3.

✓ Tipp:

- Man kann die ID-Regeln auch mit der lokalen Methode, also mit dem Attribut `style` direkt im Tag angeben. Dadurch wird der Quelltext aber schnell unübersichtlich. Daher bevorzuge ich die interne Methode für die Objekt-Definition.

```
<html><head><title>Objekte</title>
<style type="text/css">
<!--
#objekt1 { position:absolute; top:40px;
left:60px; width:150px; }
h2 { font:bold 16px Arial,sans-serif;
color:#FF0000; }
-->
</style></head>
<body>
<div id="objekt1">
<h2>Auch diese Schrift gehört zu Objekt1
</h2>
<img src="b.gif" width="35" height="32"
border="0">
</div>
</body></html>
```

Listing 11.1: Ein erzeugtes Objekt

Abbildung 11.3: Browseransicht

Die Bedeutung der Event-Handler	
HANDLER	**DAS EREIGNIS TRITT EIN, WENN...**
onclick	...das Element angeklickt wird.
onmouseover	...man mit der Maus über das Element fährt.
onmouseout	...man mit der Maus über das Element gefahren ist und dieses nun verlässt.
onmousedown	...man die Maustaste gedrückt hält.
onmouseup	...man die Maustaste loslässt.
onkeypress	...man eine Keyboard-Taste gedrückt hält.
onload	...das Element geladen wurde.
onunload	...die Datei verlassen wird.
onfocus	...ein Element den Fokus erhält (aktiviert wird).
onblur	...ein Element den Fokus verliert. (Das Element war aktiviert und wird nun verlassen.)

Tabelle 11.2: Gebräuchliche Event-Handler, die ab IE4 und N4 unterstützt werden

Was sind Event-Handler?

Event-Handler (Ereignis-Behandler) sind das Bindeglied zwischen HTML und einer JavaScript-Aktion, die ausgeführt werden soll.

Ein Event (Ereignis) findet auf einer Webseite z.B. dann statt, wenn der Anwender mit der Maus auf ein Element klickt. Sobald das Ereignis eintritt, führt der Event-Handler, der als Attribut in dem Element-Tag notiert wird, die ihm zugeordnete Aktion aus.

Es gibt etliche verschiedene Event-Handler; Sie erkennen sie daran, dass sie mit `on` beginnen, z.B. `onclick`.

In Tabelle 11.2 finden Sie die gebräuchlichsten Event-Handler, die ab IE4 und N4 unterstützt werden.

Netscape 4 und Event-Handler

Nicht jeder Event-Handler kann auf jedes Element angewendet werden, und nicht alle Event-Handler werden von jedem Browser unterstützt.

Netscape 4 schränkt uns bei der Anwendung von Event-Handlern stark ein. Dieser Browser lässt sie grundsätzlich nur in Formularelementen und den Tags `<body>`, `<frameset>`, `<a>`, `<img>` und `<area>` (Imagemap-Bereiche) zu. Bei Formularen macht Netscape je nach Event-Handler zusätzlich Unterschiede bei den Formularelementen.

Das macht uns das Leben nicht gerade einfach. Da Netscape 4 aber immer noch sehr oft eingesetzt wird, ist dies der kleinste gemeinsame Nenner, nach dem wir uns richten müssen.

In Tabelle 11.3 finden Sie noch einmal die gebräuchlichsten Event-Handler, diesmal mit den Angaben, in welchen Tags sie bei Netscape erlaubt sind. Die angegebenen Objekte entsprechen den Bezeichnungen aus der Objekt-Hierarchie (siehe Abbildung 11.1 im Abschnitt *Was ist das DOM?)*. Ausschlaggebend sind aber die Tags, die für die Objektgruppe angegeben sind. Handler, die in gleichen Tags erlaubt sind, stehen in einer Zelle untereinander.

Einsatz der Event-Handler in Netscape 4

HANDLER	OBJEKTE	HTML-TAGS
onclick	images links forms	<img> <area> <a> <input> <textarea>
onmouseover onmouseout	links images	<a> <area> nur für Imagemap-Bereiche
onmousedown onmouseup	links forms	<a> <input type="button"> nur für Buttons
onkeypress	forms	<input> <textarea>
onload onunload	window	<body> <frameset>
onfocus onblur	window forms	<body> <frameset> <frame> <input> <select> <textarea>

Tabelle 11.3: Eventhandler in Netscape 4

✓ Tipp:

- Bei Event-Handlern, die im `<a>`-Tag erlaubt sind, können Sie Netscape austricksen. Mehr dazu auf der nächsten Seite.

```
<html><head>
<title>Netscape ausgetrickst</title>
<style type="text/css">
<!--
p { font:bold 15px Arial; color:#000000;
text-align:justify; }
h2 { font:bold 20px Arial; color:#FF0000;
}
-->
</style></head>
<body>
<h2><a href=""
style="text-decoration:none;
color:#FF0000;"
onclick="alert('Netscape
ausgetrickst!');return false;">
Klick mich an!
</a></h2>
</body></html>
```

Listing 11.2: Netscape ausgetrickst

Abbildung 11.4: Netscape wird durch einen leeren Link ausgetrickst.

Netscape 4 austricksen

Es gibt einen Weg, wie Sie Netscape 4 häufig austricksen können. Da für einige Event-Handler die Anwendung in Hyperlinks erlaubt ist, können Sie um ein Element oder eine Textpassage einen leeren Link legen. Leer bedeutet, dass kein Wert beim `href`-Attribut angegeben wird, auch nicht das Rautezeichen #.

Das Problem bei diesem Trick ist die browsertypische Link-Auszeichnung. Sie kann aber bei Bedarf entfernt werden. Dazu weist man dem Link die Stildefinitionen `text-decoration:none;` und `font:[wie Text];` zu.

Für den Event-Handler `onclick` muss außerdem der Wert `false` zurückgegeben werden, sonst wird das `onclick`-Ereignis verarbeitet. Da der Link aber auf eine leere URL verweist, würde dann je nach Konfiguration des Webservers eine Fehlermeldung oder der Inhalt des Verzeichnisses, in dem das Dokument enthalten ist, angezeigt werden.

Ein Beispiel finden Sie in Listing 11.2. Wenn man die Überschrift anklickt, erscheint eine Meldung mit dem Text "Netscape ausgetrickst!" (siehe Abbildung 11.4).

Event-Handler einsetzen

Der Mausklick auf ein bestimmtes Element ist eine Aktion, die vom Event-Handler erkannt wird. Der Handler löst daraufhin eine bestimmte Reaktion aus. Diese Informationen müssen miteinander verknüpft werden.

So setzen Sie einen Event-Handler ein:

1. `<a href="seite2.html"`
   Beginnen Sie mit dem Tag für das Element, dem der Event-Handler hinzugefügt werden soll.
2. `onmouseover=`
   Der gewünschte Event-Handler wird als Attribut notiert. Direkt nach dem Handler setzen Sie ein Gleichheitszeichen, das den nachfolgenden Wert einleitet.
3. `"Funktion1();"`
   Der Wert des Event-Handlers wird in Anführungszeichen geschrieben. Das kann eine Funktion sein, oder auch ein direkter Befehl. Beispiele: `"alert('Hallo Welt!');"` `"document.images.b2.src='d.gif'"`
4. `>Inhalt</a>`
   Schließen Sie das Start-Tag, schreiben Sie den Inhalt des Elements und setzen Sie dann das End-Tag.

Listing 11.3 zeigt ein Beispiel für einen Event-Handler. In Abbildung 11.5 und Abbildung 11.6 sehen Sie, was geschieht, wenn man mit der Maus über den Link fährt.

✓ Tipp:

- Wenn Werte wie "bild.gif" Bestandteil eines Handler-Wertes sind, müssen sie in einfache Anführungszeichen gesetzt werden. Doppelte Anführungszeichen würden zu JavaScript-Fehlern führen.

```
<html><head>
<title>Event-Handler</title>
<style type="text/css"><!--
p { font:15px Arial; color:#000000;
text-align:justify; }
--></style></head>
<body>
<p>Wenn man mit der Maus über den Link
fährt, ändert sich die Grafik.
<a href="#" onmouseover="document.images.
bild1.src='d.gif'">
Mouseover-Effekt:</a></p>
<p>
<img name="bild1" src="b.gif" width="35"
height="32" border="0">
</p>
</body></html>
```

Listing 11.3: Bitte beachten Sie, dass die Zeilenumbrüche im `<a>`-Tag bzw. im Wert des Event-Handlers durch die Spaltenbreite im Buch bedingt ist. Im Quelltext der Seite sind diese Umbrüche nicht vorhanden.

Abbildung 11.5: Die Grafik ändert sich…

Abbildung 11.6: …wenn man mit der Maus über den Link fährt.

```
<p>Wenn man mit der Maus über den Link
fährt, ändert sich die Grafik.
<a href="#" onmouseover="document.images.
bild1.src='c.gif'"
onmouseout="document.images.
bild1.src='d.gif'">
Mouseover-Effekt:</a></p>
<p>
<img name="bild1" src="b.gif" width="35"
height="32" border="0">
</p>
```

Listing 11.4: Ein Auszug aus dem Quelltext der Seite. Das `<a>`-Tag wurde um den Event-Handler `onmouseout` erweitert.

Abbildung 11.7: Nach dem Laden der Seite

Abbildung 11.8: `Onmouseover`

Abbildung 11.9: `Onmouseout`

Mehrere Event-Handler verwenden

Sie können in einem Element auch mehrere Event-Handler verwenden. Im vorigen Beispiel bietet es sich an, die Ursprungsgrafik wieder anzuzeigen, wenn der Maus-Cursor den Link verlässt.

So wenden Sie mehrere Event-Handler in einem Element an:

1. `<a href="seite2.html"`
   Beginnen Sie mit dem Tag für das Element, dem die Event-Handler hinzugefügt werden sollen.
2. `onmouseover="Funktion1();"`
   Notieren Sie den ersten Event-Handler mit seinem Wert und geben Sie dahinter ein Leerzeichen ein.
3. `onmouseout="Funktion2();"`
   Notieren Sie den nächsten Event-Handler mit seinem Wert und geben Sie dahinter ein Leerzeichen ein, wenn weitere Event-Handler folgen sollen.
4. `>Inhalt</a>`
   Notieren Sie weitere Event-Handler wie in Schritt 3. Nach dem letzten Handler schließen Sie das Start-Tag, schreiben den Inhalt des Elements und setzen dann das End-Tag.

In Listing 11.4 sehen Sie einen Auszug aus dem Quelltext. Es wurde der gleiche Code verwendet wie in Listing 11.3, lediglich das `<a>`-Tag wurde um den Event-Handler `onmouseout` erweitert. In Abbildung 11.7, Abbildung 11.8 und Abbildung 11.9 sehen Sie, was geschieht, wenn man mit der Maus über den Link fährt und ihn wieder verlässt. Um den Effekt zu verdeutlichen, wurde eine dritte Grafik verwendet.

Ein Event-Handler mit mehreren Werten

Sie können in einem Event-Handler auch mehrere Werte verwenden. Damit lösen Sie mehrere Reaktionen auf ein Ereignis gleichzeitig aus.

So wenden Sie mehrere Werte in einem Event-Handler an:

1. `<a href="#"`
   Beginnen Sie mit dem Tag für das Element, dem der Event-Handler hinzugefügt werden soll.
2. `onmouseover=`
   Notieren Sie den Event-Handler und direkt anschließend ein Gleichheitszeichen.
3. `"Funktion1();`
   Setzen Sie ein öffnendes Anführungszeichen und notieren Sie den ersten Wert, gefolgt von einem Semikolon.
4. `Funktion2();Funktion3()"`
   Notieren Sie weitere Werte; sie werden jeweils durch ein Semikolon voneinander getrennt. Nach dem letzten Wert setzen Sie das schließende Anführungszeichen.
5. `>Inhalt</a>`
   Das Start-Tag wird geschlossen, dann schreiben Sie den Inhalt des Elements und setzen das End-Tag.

In Listing 11.5 sehen Sie ein Beispiel. Das Ergebnis im Browser zeigen Abbildung 11.10 und Abbildung 11.11.

```
<html><head>
<title>Event-Handler</title>
<style type="text/css"><!--
p { font:15px Arial; color:#000000;
text-align:justify; }
--></style></head>
<body>
<p><a href="#" onmouseover=
"document.images.bild1.src='c.gif';
document.images.bild2.src='d.gif';
document.images.bild3.src='b.gif'">
Alle Grafiken ändern:</a></p>
<p>
<img name="bild1" src="b.gif" width="35"
height="32" border="0">
<img name="bild2" src="c.gif" width="35"
height="32" border="0">
<img name="bild3" src="d.gif" width="35"
height="32" border="0">
</p>
</body></html>
```

Listing 11.5: Event-Handler mit mehreren Werten

Abbildung 11.10: Die drei Grafiken werden gleichzeitig geändert…

Abbildung 11.11: …wenn man mit der Maus über den Link fährt.

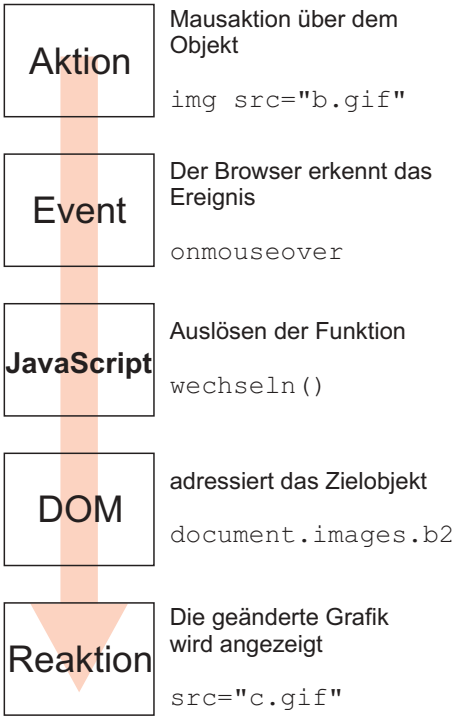

Abbildung 11.12: Ein Ereignis mit JavaScript

So funktioniert das DOM

Nachdem wir uns nun mit Objekten und Event-Handlern beschäftigt haben, schauen wir uns an, wie das DOM arbeitet. Sie haben das DOM schon oft in Aktion gesehen, ohne dass es Ihnen wahrscheinlich bewusst war. Fast überall, wo JavaScript und DHTML eingesetzt werden, ist das DOM am Werk.

In Abbildung 11.12 sehen Sie, was im Einzelnen geschieht, wenn auf einer Webseite mit JavaScript ein Ereignis eintritt. Sie bezieht sich auf Listing 11.6.

```
<html><head>
<title>So funktioniert das DOM</title>
<script language="JavaScript">
<!--
function wechseln() {
    document.images.b2.src="c.gif"
    }
//-->
</script></head>
<body> <a href="seite2.html"
onmouseover="wechseln()">
<img src="b.gif" name="b2" width="35"
height="32" border="0">
</a>
</body></html>
```

Listing 11.6: Funktionsweise des DOM

Die Unterschiede der DOMs

Es wurde bereits mehrfach erwähnt: Wir sehen uns verschiedenen DOMs gegenüber, da Netscape 4 und Internet Explorer 4 noch ihre eigenen DOM-Technologien verwenden, während ab N6 und IE5 das DOM des W3C integriert ist.

Bei den Beispielen für Event-Handler wurde mit Grafiken gearbeitet. Diese Beispiele funktionieren in Netscape 4 genauso wie in IE und Opera. Wo liegt also das Problem?

Zugriff auf positionierte Objekte

Die Eigenschaften eines positionierten Objekts zu ändern ist etwas anderes, als eine Bildquelle auszutauschen. Für beides wird ein DOM benutzt, aber das DOM für positionierte Objekte ist anders und je nach Browser verschieden.

Man spricht im Allgemeinen von Ebenen, wenn es um positionierte Objekte geht. Netscape 4 versteht z.B. zwar das `<div>`-Tag, mit dem Ebenen gesetzt werden. N4 lässt aber nicht zu, die Eigenschaften eines solchen `<div>`-Bereichs mit JavaScript zu beeinflussen, da dieser Browser seine eigene Ebenen-Technologie hat, ebenso wie der Internet Explorer. Den Zugriff auf diese Ebenen regeln je nach Browser verschiedene DOMs:

- `Layer-DOM` (Netscape)
- `All-DOM` (Internet Explorer)
- `ID-DOM` (W3C, alle neueren Browser)

Welcher Browser welches DOM unterstützt, finden Sie in Tabelle 11.4.

DOMs der DHTML-fähigen Browser

BROWSER	VERSION	DOM
Netscape	4	Layer
	6	ID
Internet Explorer	4	All
	5	All, ID
	6	All, ID
Opera	4	ID
	5	ID
	6	ID

Tabelle 11.4: Die DOMs der verschiedenen Browserversionen

```
<html><head>
<title>Layer-DOM</title>
<script language="JavaScript">
<!--
function unten() {
  document.layers['Absatz1'].top=40;
  }
function oben() {
  document.layers['Absatz1'].top=10;
  }
//--></script>
<style type="text/css"><!--
#Absatz1 { position:relative; top:10px;
left:10px; }
--></style></head>
<body><p>
<a href="#" onclick="unten();">
<b>Absatz1 nach unten</b></a>  
<a href="#" onclick="oben();">
<b>Absatz1 nach oben</b></a></p>
<div id="Absatz1">Dieses Objekt hat die ID
"Absatz1". </div>
</body></html>
```

Listing 11.7: Das Layer-DOM und sein Einsatz

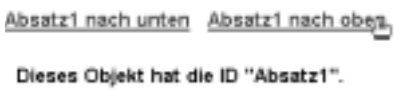

Abbildung 11.13: Die Seite nach dem Laden oder wenn man den rechten Link anklickt

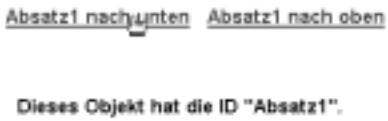

Abbildung 11.14: Der Absatz wurde durch Anklicken des Links verschoben.

Das Layer-DOM (Netscape 4)

DHTML beruht bei Netscape4 auf der Layer-Technik. Diese Layer sind ebenfalls Ebenen. Um Verwechslungen mit den CSS-Ebenen vorzubeugen spricht man dabei von Netscape-Ebenen, während der Begriff Ebenen für die CSS-Ebenen steht.

Die Netscape-Ebenen können durch die Tags `<layer>` oder `<div>` gebildet werden. Da andere Browser das `<layer>`-Tag nicht unterstützen, werden in der Praxis die `<div>`-Tags eingesetzt.

Bei Netscape 4 ist es nicht möglich, nachträglich auf beliebige Elemente zuzugreifen. N4 lässt nur Änderungen an Ebenen zu. Man kann z.B. die Formateigenschaften eines `<h2>`-Tags mit JavaScript nicht ändern.

Das Layer-DOM lässt direkte Änderungen an der Sichtbarkeit, der Position und dem sichtbaren Bereich eines Objekts zu. Diese Änderungen werden ohne erneutes Laden der Seite angezeigt. Andere Änderungen (z.B. Textfarbe, Schriftart, Ränder usw.) erfordern das Neuladen einer Seite, um angezeigt zu werden.

In Listing 11.7 wurde ein Objekt mit der ID "Absatz1" erstellt. Um für dieses Objekt die Position von oben zu ändern, muss über das folgende DOM zugegriffen werden:

```
document.layers['Absatz1'].top
```

Das Beispiel funktioniert nur in Netscape 4. In Abbildung 11.13 und Abbildung 11.14 sehen Sie das Layer-DOM in Aktion.

Das All-DOM (Internet Explorer 4)

DHTML beruht bei Internet Explorer 4 auf der All-Technik. Es ist möglich, nachträglich auf alle Elemente zuzugreifen.

Das All-DOM lässt direkte Änderungen an allen Eigenschaften eines Objekts zu, sofern der IE diese Eigenschaften interpretiert. Diese Änderungen werden ohne erneutes Laden der Seite angezeigt.

In Listing 11.8 wurde ein Objekt mit der ID "Ab1" erstellt. Um für dieses Objekt die Position von oben zu ändern, muss über das folgende DOM zugegriffen werden:

```
document.all['Ab1'].style.top
```

In Abbildung 11.15 und Abbildung 11.16 sehen Sie das All-DOM in Aktion. Das Beispiel funktioniert nur im IE.

```
<html><head>
<title>All-DOM</title>
<script language="JavaScript">
<!--
function unten() {
document.all['Ab1'].style.top=40;
  }
function oben() {
document.all['Ab1'].style.top=10;
  }
//--></script>
<style type="text/css"><!--
#Ab1 { position:relative; top:10px;
left:10px; }
 --></style></head>
<body><p>
<a href="#" onclick="unten();">
<b>Ab1 nach unten</b></a>  
<a href="#" onclick="oben();">
<b>Ab1 nach oben</b></a></p>
<div id="Ab1">Dieses Objekt hat die ID
"Ab1". </div>
</body></html>
```

Listing 11.8: Das All-DOM in Aktion

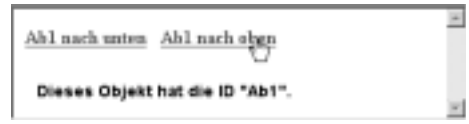

Abbildung 11.15: Die Seite nach dem Laden oder wenn man den rechten Link anklickt

Abbildung 11.16: Der Absatz wurde durch Anklicken des Links verschoben.

```
<html><head>
<title>ID-DOM</title>
<script language="JavaScript">
<!--
function unten() {
document.getElementById('Ab1').style
.top=30;
  }
//--></script>
<style type="text/css"><!--
#Ab1 { position:relative; top:0px;
left:10px; }
 --></style></head>
<body><p>
<a href="#" onclick="unten();">
<b>Ab1 nach unten</b></a> </p>
<div id="Ab1">Dieses Objekt hat die ID
"Ab1". </div>
</body></html>
```

Listing 11.9: Beachten Sie, dass der Zeilenumbruch in der Funktion unten() nur durch die Spaltenbreite im Buch bedingt ist. Im tatsächlichen Quelltext ist der Umbruch nicht vorhanden.

Abbildung 11.17: Die Seite nach dem Laden

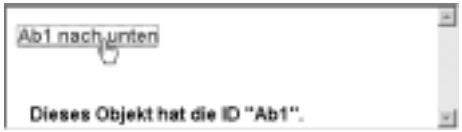

Abbildung 11.18: Der Absatz wurde durch Anklicken des Links verschoben.

Das ID DOM des W3C

Das DOM, das vom W3C entwickelt wurde, wird als Standard-Dom oder ID-Dom bezeichnet. Dieses DOM wird in neueren Browserversionen integriert. IE benutzt das ID-DOM seit der Version 5, Opera seit Version 4, Netscape erst in der neuesten Version 6.

DHTML beruht in diesem DOM auf der Technik `getElementById`. Die Objekte werden damit anhand ihrer ID identifiziert. Es ist möglich, nachträglich auf alle Elemente zuzugreifen.

Das ID DOM lässt direkte Änderungen an allen Eigenschaften eines Objekts zu. Diese Änderungen werden ohne erneutes Laden der Seite angezeigt.

In Listing 11.9 wurde wieder ein Objekt mit der ID "Ab1" erstellt. Um für dieses Objekt die Position von oben zu ändern, muss über das folgende DOM zugegriffen werden:

```
document.getElementById('Ab1').style.top
```

In Abbildung 11.17 und Abbildung 11.18 sehen Sie das ID-DOM in Aktion. Das Beispiel funktioniert in allen Browsern, die das ID-DOM unterstützen.

Achtung!

Bei `getElementById` ist die Schreibweise des Begriffs (Groß-/Kleinschreibung) wichtig. Wenn Sie sich vertippen, funktioniert das DOM nicht! Beachten Sie auch, das die Objekt-ID hier nicht in eckigen, sondern in runden Klammern steht.

Die Problematik der drei DOMs

Die Beschreibung der drei verschiedenen DOMs macht es deutlich: Um mit allen gängigen Browsern auf ein bestimmtes Objekt zugreifen zu können, wird für jedes DOM eine andere Funktion benötigt.

Wie kann man diesem Problem begegnen? Theoretisch wäre es möglich, alle Funktionen in mehrfacher Ausführung zu programmieren und durch eine Browserabfrage den Zugriff entsprechend zu steuern. Wenn es um einen einzigen Objekt-Zugriff geht, ist das akzeptabel. Wenn es aber um mehrere Funktionen geht, ist der Arbeitsaufwand nicht mehr zu vertreten.

Die Lösung des Problems

Wir programmieren ein Script, das uns die Arbeit abnimmt und – als extern gespeicherte Datei – immer wieder eingesetzt werden kann. Wir erschaffen uns ein so genanntes CrossBrowser-DOM. Wie man das macht, lernen Sie im nächsten Kapitel.

12

DAS CROSSBROWSER-DOM

Das letzte Kapitel hat es deutlich gezeigt: Wenn wir browserübergreifendes DHTML schreiben wollen, ohne jede einzelne Funktion in dreifacher Ausführung programmieren zu müssen, brauchen wir eine Lösung, die universell einsetzbar ist.

In diesem Kapitel lernen Sie Schritt für Schritt, wie man eine solche Lösung programmiert. Der Einfachheit halber nennen wir unser Programm CrossBrowser-DOM, weil es in der Praxis so arbeitet. Natürlich ist das Script nicht wirklich ein DOM, denn ein solches gehört, wie wir wissen, zur Browsertechnologie.

Abfrage des DOM-Typs

Das CrossBrowser-DOM, das wir erstellen werden, muss ein Objekt automatisch richtig adressieren, unabhängig davon, welches der drei „echten“ DOMs der Browser verwendet.

Dazu brauchen wir eine Funktion, die je nach verwendetem DOM entsprechend reagiert. Solche Scripts nennt man im Fachjargon auch *Weiche*.

Man könnte eine Weiche schreiben, die den verwendeten Browser identifiziert. Das allein reicht aber nicht, da die Browser je nach Version verschiedene DOMs verwenden.

Sinnvoller ist es, zuerst den Fähigkeiten des Browsers entsprechend die passende Funktion aufzurufen. Dazu fragen wir den DOM-Typ ab, der verwendet wird. Mit einigen Zeilen JavaScript ist das schnell erledigt.

In Listing 12.1 sehen Sie eine einfache Browserabfrage, die testet, ob der Browser das ID-DOM kennt.

```
<html><head>
<title>Browserabfrage</title>
</head>
<body>
<script language="JavaScript"><!--
if (document.getElementById) {
    document.write("Ich verstehe das ID
DOM<br><br>");
    }
else {
    document.write("Das ID-DOM verstehe
ich nicht!<br>");
    }
//--></script>
</body></html>
```

Listing 12.1: Die Abfrage testet, ob der Browser das ID-DOM versteht.

Abbildung 12.1: Operas Antwort auf die Abfrage

Abbildung 12.2: Auch der Internet Explorer 5 versteht das ID-DOM.

Abbildung 12.3: Netscape 4 kennt das ID-DOM nicht.

Die einfache Browserabfrage

Die wenigen JavaScript-Befehle in Listing 12.1 müssten eigentlich klar sein.[1] Der Aufbau wird hier dennoch erklärt:

1. `if (document.getElementById)` ist die Bedingung, mit der getestet wird, ob der Browser das ID-DOM versteht.
2. `{ document.write("…Ja…"); }` ist die Anweisung des `if`-Blocks, die ausgeführt wird, wenn der Browser das ID-DOM versteht.
3. `else` kommt zum Einsatz, wenn der Browser das ID-DOM nicht versteht.
4. `{ document.write("…Nein…"); }` Auch der `else`-Block hat eine Anweisung. Wenn der Browser das ID-DOM nicht versteht, wird diese Anweisung ausgeführt.

In Abbildung 12.1, Abbildung 12.2 und Abbildung 12.3 sehen Sie, wie die Browser auf diese Abfrage reagieren.

1. JavaScript-Kenntnisse sind Voraussetzung für DHTML. Einen leichten Einstieg in das Thema JavaScript bietet Ihnen das Buch *JavaScript*, ISBN 3-8272-5960-6 aus der Buchreihe *Digital Studio One*.

Alle DOM-Typen abfragen

Wir benötigen eine Abfrage, die alle drei DOM-Typen einschließt. Für unser CrossBrowser-DOM brauchen wir außerdem Werte, die wir weiterverarbeiten können.

Dazu erstellen wir für jeden DOM-Typ eine Variable und zusätzlich eine Variable für die DHTML-Fähigkeit des Browsers. Mit dieser Variablen kann man z.B. Browser, die keines der drei DOMs kennen, auf Alternativseiten umleiten. Die vier Variablen werden mit dem Wert `0` (`false`) initialisiert.

Für die Abfrage eignet sich eine verschachtelte `if-else`-Verzweigung. Wenn ein DOM erkannt wird, werden die anderen DOMs nicht mehr abgefragt. Für das erkannte DOM wird der entsprechende Variablen-Wert auf `1` (`true`) gesetzt, ebenso der Wert für die DHTML-Fähigkeit. Mit den Rückgabewerten kann man dann weiterarbeiten.

In Listing 12.2 sehen Sie das Script für die erweiterte Browserabfrage. Die Verschachtelung ist durch die Einrückungen gut zu erkennen. Für das ID-DOM wurden die Befehle hervorgehoben. Aus Platzgründen wurde nur der Head-Bereich der Seite im Listing dargestellt.

```
<html><head>
<title>Erweiterte Browserabfrage</title>
<script language="JavaScript">
<!--
var idDOM = 0; //ID-DOM?
var allDOM = 0; //All-DOM?
var layDOM = 0; //Layer-DOM?
var DHTML = 0; //DHTML-fähig?


if (document.getElementById) {
 idDOM = 1; DHTML = 1;
}
else {
    if (document.all) {
    allDOM = 1; DHTML = 1;
    }
    else {
        if (document.layers) {
        layDOM = 1; DHTML = 1; }
        }
    }

//--></script>
<head>
```

Listing 12.2: Das Script prüft zuerst, ob der Browser das ID-DOM kennt. Wenn das der Fall ist, werden die Variablen `idDOM` und `DHTML` auf `1` (`true`) gesetzt. Wenn der Browser das ID DOM nicht kennt, wird die `else`-Abfrage ausgeführt, in der zuerst auf das All-DOM geprüft wird. Kennt der Browser auch dieses nicht, prüft er schließlich auf das Layer-DOM. Bei einem Browser, der keines der DOMs kennt, wird keine der Variablen auf `1` (`true`) gesetzt.

```
<body>
<script language="JavaScript">
<!--
if (DHTML) {
document.write("<br>Das ist ein DHTML-
fähiger Browser.<br><br>Er benutzt das ");
}

if (idDOM) {
document.write("ID DOM. ");
}
else {
if (allDOM) {
document.write("All DOM. ");
}
else {
if (layDOM) {
document.write("Layer DOM. "); }
else {
document.write("Dieser Browser kann mit
DHTML nicht umgehen. ");
} } }
//-->
</script>
</body></html>
```

Listing 12.3: Erweiterte Browserabfrage

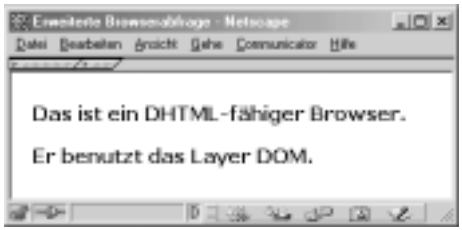

Abbildung 12.4: Die Anzeige in Netscape 4

Ein Beispiel für den Einsatz der erweiterten Abfrage:

Damit Sie die Wirkungsweise der erweiterten Browserabfrage beobachten können, wurde im Beispiel (siehe Listing 12.3) noch einmal mit Textausgaben gearbeitet. Auch im Ausgabe-Script wurde mit verschachtelten `if-else`-Verzweigungen gearbeitet. (Wegen der begrenzten Spaltenbreite im Buch wurde hier auf die Einrückungen verzichtet.)

1. Das Script prüft mit `if (DHTML)` zuerst, ob der Rückgabewert für die Variable DHTML `1` (true) ist. Das ist nur der Fall, wenn der Browser eines der DOMS kennt. Wenn das nicht der Fall ist, wird die `else`-Anweisung ausgeführt, die dem Besucher mitteilt, dass der Browser nicht DHTML-fähig ist.
2. Ist der Browser DHTML-fähig, wird der entsprechende Text ausgegeben. Je nachdem, für welche DOM-Variable der Rückgabewert auf `true` gesetzt ist, wird der Text dann durch Ausgabe des DOM-Typs ergänzt.

Aus Platzgründen fehlt im Listing der Head-Bereich, in dem das Script mit der erweiterten Browserabfrage steht. Der komplette Quelltext für die Seite in Abbildung 12.4 besteht aus Listing 12.2 und Listing 12.3.

Wieder ein Netscape-Bug

Damit wir das CrossBrowser-DOM in diversen Webseiten einsetzen können, werden wir das Programm als externe Datei speichern, die dann im Head der Seiten referenziert wird. Das spart nicht nur Schreibarbeit, sondern auch Ladezeit und Speicherplatz auf dem Webserver.

In Netscape 4 funktioniert aber unter bestimmten Umständen die Abfrage des DOM-Typs nicht. Dieser Bug tritt nur auf, wenn das Script als externe Datei gespeichert ist, und nur bei der Abfrage des DOM-Typs. Wenn eine Abfrage nach dem Browsertyp als externes Script gespeichert ist, hat Netscape damit keine Probleme, also ändern wir unsere Abfrage für das Layer-DOM entsprechend ab.

In Listing 12.4 sehen Sie die geänderte Abfrage für das Layer-DOM. Die Erläuterung der `else`-Anweisung spare ich mir an dieser Stelle. Mehr zum Identifizieren eines Browsers erfahren Sie in Kapitel 13.

Das geänderte Script kann nun als externe Datei verwendet werden, ohne dass in Netscape4 ein Bug auftritt. Die Dateiendung muss *.js* lauten. Das externe Script wird in den folgenden Beispielen im Head der Webseiten referenziert. Ab sofort sehen Sie in den Listings zum CrossBrowser-DOM also nur noch den reinen JavaScript-Code. Die Datei wurde als *dom.js* gespeichert.

```
var idDOM = 0; //ID-DOM?
var allDOM = 0; //All-DOM?
var layDOM = 0; //Layer-DOM?
var DHTML = 0; //DHTML-fähig?


if (document.getElementById) {
 idDOM = 1; DHTML = 1;
}
else {
if (document.all) {
allDOM = 1; DHTML = 1;
}
else {
browserVersion = parseInt(navigator.app-
Version);
if ((navigator.appName.indexOf
('Netscape') != -1) &&
(browserVersion == 4)) {
layDOM = 1; DHTML = 1; }
}
}
//Ende des Scripts
```

Listing 12.4: Beachten Sie, dass die Zeilenumbrüche in der `if`-Anweisung durch die Spaltenbreite im Buch bedingt sind. Im Code der Datei sind sie nicht vorhanden.

Abbildung 12.5: Die Anzeige in N4 sieht nach der Änderung der Abfrage genauso aus wie vorher.

```
function crossDOM(Stil,ObjID) {

if (idDOM) {
    if(Stil == 1) {
    return (document.
getElementById(ObjID).style);
    }
    else {
    return (document.
getElementById(ObjID));
    }
}

if (allDOM) {
    if(Stil == 1) {
    return (document.
all[ObjID].style);
    }
    else {
    return (document.all[ObjID]);
    }
}

else {
    if (layDOM) {
    return (document.layers[ObjID]);
    }
}

}
//Ende der Funktion crossDOM
```

Listing 12.5: Die Grundfunktion des CrossBrowser-DOMs

Die DOM-Funktion

Nun kommen wir zur Grundfunktion des Scripts. Wir erweitern die Datei *dom.js* durch eine Funktion, mit der die Adressierung der Objekte je nach „echtem“ DOM umgewandelt wird.

Die Funktion heißt `crossDOM` und es werden zwei Parameter übergeben: `ObjID` und `Stil`. `ObjID` steht für den ID-Namen der entsprechenden Objekte, auf die wir zugreifen wollen.

Der Parameter `Stil` wird benötigt, weil sowohl im All-DOM als auch im ID-DOM der Zusatz `.style` verwendet wird, wenn auf Stil-Eigenschaften eines Objekts zugegriffen werden soll. Als Beispiel für einen solchen Zugriff sehen Sie nachfolgend die drei DOMs für die `top`-Position des Objekts "Abs2":

```
document.layers['Abs2'].top
document.all['Abs2'].style.top
document.getElementById('Abs2').style.top
```

Es muss also die Möglichkeit geschaffen werden, die Objektadresse sowohl mit dem Zusatz `.style` als auch ohne anzugeben. Dazu wurde eine Verzweigung mit der Bedingung `if(Stil == 1)` ins Script eingefügt. Wenn für den Parameter `Stil` der Wert `1` übergeben wird, wird die DOM-Adresse mit dem Zusatz `.style` versehen.

In Listing 12.5 sehen Sie die Funktion. Aufgrund der Spaltenbreite im Buch sehen Sie in einigen der `return`-Anweisungen Umbrüche, die im tatsächlichen Code nicht vorhanden sind.

Einsatz des CrossBrowser-DOMs

Das CrossBrowser-DOM wird im Laufe dieses Kapitels zwar noch um einige Funktionen erweitert; es ist aber bereits einsatzfähig.

So wird das DOM verwendet:

1. `<script src="dom.js></script>`
   Mit diesem Befehl wird das Script in die Webseite eingebunden.
2. `function pos(ObjID,oben,links)`
   Als Beispiel wird in einem weiteren <script>-Bereich eine Funktion verwendet, die Objekte verschieben kann; sie wurde `pos` genannt. Es werden drei Parameter übergeben: die Objekt-ID und die Positionen `top` und `left`.
3. `neuePos = crossDOM(1,ObjID)`
   Die Variable `neuePos` schreibt die DOM-Adresse. Da die Position eines Objekts im ID DOM und All DOM mit `.style` adressiert wird, muss der Funktion `crossDOM` für den Parameter `Stil` der Wert `1` übergeben werden. Wenn das Objekt die ID "b1" hat und der Browser das All-DOM verwendet, erhält die Variable `neuePos` also den Wert `document.all['b1'].style`

Fortsetzung auf der nächsten Seite

```
<html><head>
<title>Das CrossBrowser-DOM</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function pos(ObjID,oben,links) {
    neuePos = crossDOM(1,ObjID) ;
    neuePos.top = oben ;
    neuePos.left = links ;
}
//--></script>

<style type="text/css">
body { font:bold 14px Verdana;
margin-left:10px; color:#000000; }
#bild1 { position:absolute;
top:60px; left:20px; }
#bild2 { position:absolute;
top:60px; left:70px; }
</style></head>
<body><p>
<a href="#"
onClick="pos('bild1',120,20)">
Bild1 nach unten</a>
<a href="#"
onClick="pos('bild2',60,200)">
Bild2 nach rechts</a><br>
</p>
<div id="bild1"><img src="b.gif"
width="35" height="32" border="0">
</div>
<div id="bild2"><img src="c.gif"
width="35" height="32" border="0">
</div></body></html>
```

Listing 12.6: Das CrossBrowser-DOM im Einsatz

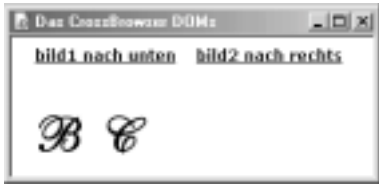

Abbildung 12.6: Die Seite nach dem Laden

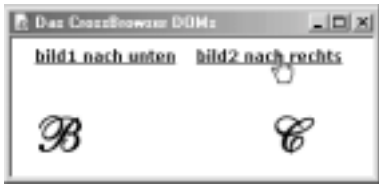

Abbildung 12.7: Durch das Anklicken des Links wird die Grafik neu positioniert.

4. `neuePos.top = oben`
   Der Variablen `neuePos.top` wird der Wert zugewiesen, der mit dem Parameter `oben` übergeben wird. Wenn dieser Wert z.B. `5` (für 5 Pixel) ist und das Objekt die ID "b1" hat, liest ein Browser mit dem `All-DOM` hier `document.all['b1'].style.top=5` aus.
5. `neuePos.left = links`
   Der Variablen `neuePos.left` wird der Wert zugewiesen, der mit dem Parameter `links` übergeben wird.
6. `#bild1 { position:absolute; top:60px; left:20px; }`
   Im `<style>`-Bereich werden die gewünschten Objekte notiert.
7. `onClick="pos('bild1',120,20)"` In der Webseite wird an der gewünschten Stelle der Event-Handler notiert, der die Funktion zum Verschieben des Objekts aufruft. Der Funktion werden hier als Parameter die Objekt-ID und die Werte für die neuen Positionen `top` und `left` übergeben.
8. `<div id="bild1"> … </div>`
   Das Objekt wird innerhalb eines `<div>`-Tags notiert. Als Attribut wird die gewünschte ID gesetzt.

Wie das Ergebnis im Browser aussieht, zeigen Abbildung 12.6 und Abbildung 12.7.

Netscape 4: eingebettete Ebenen

Das CrossBrowser-DOM funktioniert in der Version, die wir programmiert haben, in Netscape4 nur dann, wenn die Objekte nicht verschachtelt sind.

In Listing 12.7 sehen Sie ein Beispiel für verschachtelte Objekte. Das Objekt mit der ID `bild1` ist im Objekt `absatz1` eingebettet. Der Zugriff auf das Kindelement erfolgt in Netscape 4 nicht mit `document.layers['bild1']`. N4 verlangt, dass auch die Elternelemente mit angegeben werden. Die DOM-Adresse von `bild1` lautet `document.layers['absatz1'].layers['bild1']`.

Wir müssen das CrossBrowser-DOM um eine entsprechende Funktion erweitern. Man könnte die bestehende Funktion `crossDOM` abändern;, für verschachtelte Objekte müssen aber mehr Parameter übergeben werden, die für nicht verschachtelte Objekte nicht erforderlich sind. Daher empfiehlt es sich, für die verschachtelten Objekte eine neue Funktion zu schreiben.

Die neue Funktion wurde `kind` genannt und wird nur eingesetzt, wenn tatsächlich verschachtelte Objekte vorhanden sind. Als Parameter müssen außer `Stil` die Objekt-IDs für die Ebenen übergeben werden. Im Beispiel werden die Objekte nur auf einer Ebene verschachtelt, wie in Listing 12.7 gezeigt. Es werden also zwei Objekt-IDs benötigt, die im Script `ObjID1` und `ObjID2` genannt wurden (siehe Listing 12.8).

```
<div id="absatz1">
Text des ersten Absatzes
    <div id="bild1">
    <img src="b.gif">
    </div>
</div>
```

Listing 12.7: Verschachtelte Objekte

```
function kind(Stil,ObjID1,ObjID2) {

if (idDOM) {
  if(Stil == 1) {
    if (ObjID2) {
    return (document.
    getElementById(ObjID2).style);
    }
    else {
    return (document.
    getElementById(ObjID1).style);
    }
  }
  else {
    if (ObjID2) {
    return (document.
    getElementById(ObjID2));
    }
    else {
    return (document.
    getElementById(ObjID1));
    }
  }
}
```

Listing 12.8: Fortsetzung auf der nächsten Seite

Fortsetzung Listing 12.8

```
if (allDOM) {
  if(Stil == 1) {
    if (ObjID2) {
    return (document.all[ObjID2].style);
    }
    else {
    return (document.all[ObjID1].style);
    }
  }
  else {
    if (ObjID2) {
    return (document.all[ObjID2]);
    }
    else {
    return (document.all[ObjID1]);
    }
  }
}

else {
  if (layDOM) {
    if (ObjID2) {
    return (document.
    layers[ObjID1].layers[ObjID2]);
    }
    else {
    return (document.layers[ObjID1]);
    }
  }
}
}

//Ende der Funktion
```

Listing 12.8: Die Funktion für verschachtelte Objekte

Die Funktion für verschachtelte Objekte

Die Abfragen für die einzelnen DOM-Typen werden erweitert. Dabei werden die bestehenden Verzweigungen um `if (ObjID2)` erweitert und eine entsprechende `else`-Anweisung definiert, die eintritt, wenn nicht `ObjID2` angesprochen wird.

Wenn Sie in den eingebetteten Objekten weitere Objekte einbetten wollen, können Sie die Funktion `kind` jederzeit erweitern. Denken Sie bitte daran, in diesem Fall die neue Ebene auch in den Parametern der Funktion anzugeben.

✓ Tipp:

- Verschachtelungen lassen sich nicht immer vermeiden, Sie sollten aber darauf achten, nach Möglichkeit in eingebetteten Objekten keine weiteren Objekte zu verschachteln. Der Quelltext bleibt übersichtlicher und die Komplexität der Seite hält sich in einem überschaubaren Rahmen.

Einsatz der Funktion kind()

Wenn Sie verschachtelte Objekte erstellt haben, wird in der Webseite die Funktion `kind()` verwendet. Als Beispiel in Listing 12.9 wurde wieder die Funktion zum Ändern der Position eingesetzt.

In der Funktion `pos` müssen nun beide Objekt-IDs und die Werte für die neuen Positionen `top` und `left` übergeben werden. Auch im Event-Handler werden dann natürlich beide Objekt-IDs und die Werte für die Positionen eingesetzt.

```
<html><head>
<title>Das CrossBrowser-DOM</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function pos(ObjID1,ObjID2,oben,links) {
  neuePos = kind(1,ObjID1,ObjID2);
  neuePos.top = oben ;
  neuePos.left = links ;
}
//--></script>

<style type="text/css">
body { font:bold 14px Verdana;
margin-left:10px; color:#000000; }
#abs1 { position:absolute; top:20px;
left:20px; width:100% border:none; }
#bild1 { position:absolute;
top:20px; left:20px; }
#bild2 { position:absolute;
top:20px; left:70px; }
</style></head>
<body><p>
<a href="#"
onClick="pos('abs1','bild1',40,20)">
Bild1 nach unten</a>
<a href="#"
onClick="pos('abs1','bild2',20,200)">
Bild2 nach rechts</a><br>
</p>
<div id="abs1">
<div id="bild1"><img src="b.gif"
width="35" height="32" border="0">
</div>
<div id="bild2"><img src="c.gif"
width="35" height="32" border="0">
</div>
</div>
</body></html>
```

Listing 12.9: Einsatz der Funktion `kind`

13

EIGENSCHAFTEN ABFRAGEN

Für einige DHTML-Scripts ist es wichtig, die Rechnerumgebung oder bestimmte Eigenschaften eines Objekts zu kennen.

Oft geht es nur darum, den Browser zu identifizieren, manchmal will man aber auch nur wissen, wie groß das Anzeigefenster des Besuchers ist oder an welcher Stelle sich ein Element gerade befindet.

Für das Auslesen der Browsereigenschaften benötigt man das DOM nicht, zum Feststellen von Objekteigenschaften ist es erforderlich.

In diesem Kapitel lernen Sie, wie man bestimmte Eigenschaften von Objekten und Browsern herausfindet.

Allgemeines zu den Funktionen

Auf viele Funktionen, die im Lauf dieses Kapitels erstellt werden, greift man immer wieder zurück.

Sie können diese Funktionen bei Bedarf in einzelnen Webseiten notieren. Da sie aber immer wieder gebraucht werden, empfiehlt es sich, sie im CrossBrowser-DOM einzubinden. So entsteht nach und nach eine Funktionsbibliothek, die Ihnen nützliche Dienste leisten wird.

Die Namen der Funktionen können Sie so wählen, dass ein direkter Bezug zur Funktionsweise hergestellt werden kann. Das ist aber oft nur durch lange Namen möglich, was die Gefahr von Schreibfehlern erhöht. Eine Alternative ist das Durchnummerieren der Funktionen. Dann sollten Sie im Script aber in jedem Fall eine Übersicht anlegen, in der Sie die Funktionen dokumentieren (siehe Listing 13.1). Bei Bedarf können Sie die Übersicht auch ausdrucken und neben die Tastatur legen.

Kommentare im Script helfen Ihnen, sich auch nach Wochen oder Monaten noch zurecht zu finden. Diese Kommentare blähen aber auch die Scriptdatei auf und kosten damit wertvolle Ladezeit. Um das zu vermeiden, können Sie zwei Versionen des Scripts speichern. Die Datei mit den Kommentaren wird auf dem lokalen Rechner verwendet. Kopieren Sie diese Datei und entfernen Sie die Kommentare teilweise oder vollständig. Diese abgespeckte Version wird auf dem Webserver verwendet.

✓ Tipp:

- In JavaScript werden Kommentare genauso eingefügt wie in CSS (siehe Kapitel 2).

```
/*Übersicht der Funktionen:

f1: Höhe eines Objekts feststellen
Zu übergebende Parameter: Objekt-ID

f2: Breite eines Objekts feststellen
Zu übergebende Parameter: Objekt-ID

f3: Position top auslesen
Zu übergebende Parameter: Objekt-ID

f4: Position left auslesen
Zu übergebende Parameter: Objekt-ID

f5: …

Ende der Übersicht */

//Hier folgen die Funktionen

function f1(ObjID) {

// style ist nicht erforderlich

dom = crossDOM(0,ObjID)
if (dom.offsetWidth)
return dom.offsetWidth;
if (dom.clip.width)
return dom.clip.width;
return (null);
}
```

Listing 13.1: Mit einer Funktionsübersicht und Kommentaren erleichtern Sie sich die Arbeit. Ich verwende grundsätzlich die Variable `dom` für die DOM-Adresse, die mit der Funktion `crossDOM` ermittelt wird.

```
function f1(ObjID) {

// style ist nicht erforderlich

dom = crossDOM(0,ObjID)
if (dom.offsetHeight) {
return dom.offsetHeight; }
if (layDOM) {
return dom.clip.height; }
else { return (0); }
}



function f2(ObjID) {

// style ist nicht erforderlich

dom = crossDOM(0,ObjID)
if (dom.offsetWidth) {
return dom.offsetWidth; }
if (layDOM) {
return dom.clip.width; }
else { return (0); }
}
```

Listing 13.2: Die Funktionen zum Auslesen der Breite und Höhe eines Objekts. Die Datei *dom.js* wurde um diese Funktionen erweitert.

Höhe und Breite von Objekten

Jedes Objekt hat eine Breite und Höhe. Internet Explorer ab Version 4, Opera 6 und Netscape 6 verwenden zum Ermitteln der Objektbreite die Eigenschaft `offsetWidth`, Netscape 4 benutzt `clip.width`. Die Höhe wird mit `offsetHeight` bzw. `clip.height` (N4) festgestellt. Opera5 unterstützt leider keine der Eigenschaften.

Für Breite und Höhe wird je eine Funktion erstellt, in der geprüft wird, ob `offset` oder das Layer-DOM vom Browser verwendet wird. Sie finden die beiden Funktionen in Listing 13.2. Die Rückgabewerte können als Variablen weiterverwendet werden.

Der Opera-5-Bug

Die Verzweigung für Netscape4 prüft auf `layDOM`, weil Opera 5 das ID-DOM verwendet, die `offset`-Eigenschaften aber nicht unterstützt. Für die Abfrage `if(dom.clip.width)` bzw. `if(dom.clip.height)` wird in Opera 5 eine Fehlermeldung ausgegeben, weil das ID-DOM diese `clip`-Eigenschaften nicht kennt. Damit die Ausführung nicht mit einer Fehlermeldung abgebrochen wird, fragt man `if(layDOM)` ab, Opera 5 gibt als Wert dann jeweils `null` zurück.

In Opera 6 tritt dieses Problem nicht mehr auf, da hier das ID-DOM vollständig integriert ist.

Ein Beispiel für den Einsatz

In Listing 13.3 sehen Sie eine Funktion, mit der die Abmessungen des Objekts am Bildschirm ausgegeben werden.

1. `<script src="dom.js"></script>`
   Das CrossBrowser-DOM wird im Head der Seite referenziert. In dieser Datei befinden sich auch die beiden Funktionen zum Auslesen der Breite und Höhe des Objekts.
2. `function groesse(ObjID) {`
   Die Funktion erhält den Namen `groesse`. Als Parameter für die Funktion zur Ausgabe der Größe wird die Objekt-ID übergeben.
3. `hoehe = f1(ObjID);`
   `breite = f2(ObjID);`
   Es werden zwei Variablen erstellt: `hoehe` und `breite`. Diesen Variablen werden die Rückgabewerte der Funktionen `f1` bzw. `f2` zugewiesen.
4. `alert(…); }`
   Die `alert`-Anweisung gibt die Werte der beiden Variablen in einer JavaScript-Meldung aus. Danach wird die Funktion mit der schließenden Klammer beendet.

Das Ergebnis sehen Sie in Abbildung 13.1. Je nach verwendetem Browser können die ausgegebenen Werte unterschiedlich sein. Das liegt daran, dass einige Browser bei der Höhe und/oder Breite einen Rand (meistens 2px) hinzurechnen.

```
<html><head>
<title>Breite und Höhe</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function groesse(ObjID) {
hoehe = f1(ObjID);
breite = f2(ObjID) ;
alert( +breite+ 'px breit ' +hoehe+ 'px
hoch') ;
}
//--></script>
<style type="text/css">
#bild1 { position:absolute; top:40px;
left:20px; }
</style></head>
<body><p><a href="#"
onClick="groesse('bild1')">
Groesse anzeigen</a></p>
<div id="bild1"><img src="b.gif"
width="35" height="32" border="0">
</div></body></html>
```

Listing 13.3: Größe eines Objekts anzeigen

Abbildung 13.1: Anzeige der Breite und Höhe

```
function f3(ObjID) {

/* 2 Variablen erforderlich: mit und ohne
style */

dom = crossDOM(0,ObjID)
domStil = crossDOM(1,ObjID)

if (dom.offsetTop) {
return dom.offsetTop; }
if (domStil.top) {
return domStil.top; }
if (domStil.pixelTop) {
return domStil.pixelTop; }
else { return (0); }
}


function f4(ObjID) {

/* 2 Variablen erforderlich: mit und ohne
style */

dom = crossDOM(0,ObjID)
domStil = crossDOM(1,ObjID)

if (dom.offsetLeft) {
return dom.offsetLeft; }
if (domStil.left) {
return domStil.left; }
if (domStil.pixelLeft) {
return domStil.pixelLeft; }
else { return (0); }
}
```

Listing 13.4: Position top und left: Eigenschaften auslesen

Die Positionen top und left

In DHTML werden Objekte oft verschoben. Manchmal muss man dabei wissen, wo sich ein Objekt gerade befindet. Zum Auslesen der Eigenschaft `left` wird je nach Browser eine der folgenden Eigenschaften verwendet: `style.left`, `style.pixelLeft` und `offsetLeft`. Für die Position `top` sind es `style.top`, `style.pixelTop` und `offsetTop`.

Da `offsetLeft` bzw. `offsetTop` ohne `style` ausgelesen wird, benötigt man in der Funktion zwei Variablen. Eine spricht das CrossBrowser-DOM ohne `style` an, eine mit `style`.

Sie finden die beiden Funktionen in Listing 13.4.

Ein Beispiel für den Einsatz

In Listing 13.5 sehen Sie eine Funktion, mit der die Positionen des Objekts am Bildschirm ausgegeben werden.

1. `<script src="dom.js"></script>`
   Das CrossBrowser-DOM wird im Head der Seite referenziert.
2. `function PosZeigen(ObjID) {`
   Die Funktion wurde `PosZeigen` genannt. Als Parameter wird die Objekt-ID übergeben.
3. `oben = f3(ObjID);`
   `links = f4(ObjID);`
   Es werden zwei Variablen erstellt: `oben` und `links`. Diesen Variablen werden die Rückgabewerte der Funktionen `f3` bzw. `f4` zugewiesen.
4. `alert(…); }`
   Die `alert`-Anweisung gibt die Werte der beiden Variablen in einer JavaScript-Meldung aus. Danach wird die Funktion mit der schließenden Klammer beendet.

Das Ergebnis sehen Sie in Abbildung 13.2.

```
<html><head>
<title> top und left</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function PosZeigen(ObjID) {
oben = f3(ObjID);
links = f4(ObjID) ;
alert('Oben: ' +oben+ 'px, Links: '
+links+ 'px') ;
}
//--></script>
<style type="text/css">
#bild1 { position:absolute; top:40px;
left:20px; }
</style></head>
<body><p><a href="#"
onClick="PosZeigen('bild1')">
Position anzeigen</a></p>
<div id="bild1"><img src="b.gif"
width="35" height="32" border="0">
</div></body></html>
```

Listing 13.5: Ausgabe der Position

Abbildung 13.2: Anzeige der Positionen oben und links

```
/* Position unten: Höhe aus Funktion f1
und top aus Funktion f3 */

function f5(ObjID) {

hoehe = f1(ObjID) ;
oben = f3(ObjID) ;
return (hoehe + oben) ;
}



/* Position rechts: Breite aus Funktion f2
und left aus f4 */

function f6(ObjID) {

breite = f2(ObjID) ;
links = f4(ObjID) ;
return (breite + links) ;
}
```

Listing 13.6: Positionen rechts und unten

Die Positionen rechts und unten

Auch die Positionen `rechts` und `unten` können festgestellt werden, wenn auch nicht auf direktem Weg wie bei `top` und `left`. Für die Position rechts werden die Werte für links und die Breite des Objekts ermittelt und dann addiert. Für die Position unten zählt man `top` und `height` zusammen.

Für diese beiden Funktionen bräuchte man also einige Verzweigungen, wir greifen aber einfach auf die bereits bestehenden Funktionen zurück (siehe Listing 13.6).

An diesem Beispiel wird deutlich, dass eine Funktionsbibliothek in zweifacher Hinsicht Arbeit sparen kann. Zum einen kann die externe Datei in beliebigen Webseiten eingebunden werden, zum anderen können die Funktionen aufeinander zurückgreifen und sparen so zusätzliche Tipparbeit.

Ein Beispiel für den Einsatz

In Listing 13.7 sehen Sie eine Funktion, mit der die Positionen unten und rechts am Bildschirm ausgegeben werden.

1. `<script src="dom.js"></script>`
   Das CrossBrowser-DOM wird im Head der Seite referenziert.
2. `function untenrechts(ObjID) {`
   Die Funktion wurde `untenrechts` genannt. Als Parameter wird die Objekt-ID übergeben.
3. `unten = f5(ObjID);`
   `rechts = f6(ObjID);`
   Es werden zwei Variablen erstellt: `unten` und `rechts`. Diesen Variablen werden die Rückgabewerte der Funktionen `f5` bzw. `f6` zugewiesen.
4. `alert(…); }`
   Die `alert`-Anweisung gibt die Werte der beiden Variablen in einer JavaScript-Meldung aus. Danach wird die Funktion mit der schließenden Klammer beendet.

Das Ergebnis sehen Sie in Abbildung 13.3.

```
<html><head>
<title> top und left</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function untenrechts(ObjID) {
    unten = f5(ObjID);
    rechts = f6(ObjID) ;
    alert('Unten: ' +unten+
'px, Rechts: ' +rechts+ 'px') ;
}
//--></script>
<style type="text/css">
#bild1 { position:absolute; top:40px;
left:20px; }
</style></head>
<body><p><a href="#"
onClick="untenrechts('bild1')">
Position anzeigen</a></p>
<div id="bild1"><img src="b.gif"
width="35" height="32" border="0">
</div></body></html>
```

Listing 13.7: Ausgabe der Position

Abbildung 13.3: Anzeige der Positionen rechts und unten

```
/* Für Objekte den Status visible oder
hidden setzen. */

function f7(ObjID,sichtbar) {
dom = crossDOM(1,ObjID)
dom.visibility = sichtbar ;
}


/* Für Objekte ohne den Status visible
setzen */

function f8(ObjID) {
dom = crossDOM(1,ObjID)
dom.visibility = 'visible' ;
}


/* Für Objekte den Status hidden setzen */
function f9(ObjID) {
dom = crossDOM(1,ObjID)
dom.visibility = 'hidden' ;
}
```

Listing 13.8: Sie können wählen, ob Sie die universelle Funktion `f7` verwenden oder lieber die Funktionen `f8` und `f9`, in denen der Status fest vorgegeben ist. Natürlich können Sie die Funktionen auch kombinieren.

```
<body onLoad="f7('bild1','visible');
f7('bild2','hidden');f8('bild3');
f8('bild4');f9('bild5');">
```

Listing 13.9: Die Funktionen werden im `<body>`-Tag mit dem Event-Handler `onLoad` aufgerufen.

Sichtbarkeitsstatus eines Objekts

Die Sichtbarkeit eines Objekts wird in CSS mit der Eigenschaft `visibility` gesetzt (siehe Kapitel 9). Mit `visibility` wird auch der Sichtbarkeitsstatus ermittelt. Dabei sehen wir uns zwei Problemen gegenüber.

Problem 1

Einige Browser können den Status nur auslesen, wenn die Eigenschaft `dynamisch` gesetzt wird. Die CSS-Eigenschaft `visibility`, die durch CSS gesetzt wird, wird also nicht ausgelesen. Zu diesen Browsern gehört leider auch der Internet Explorer. Damit auch der IE den Sichtbarkeitsstatus auslesen kann, muss er beim Laden der Seite dynamisch gesetzt werden.

Das wird entweder mit der Funktion `f7` erledigt, mit der man den Status auf `hidden` oder `visible` setzen kann, oder mit den Funktionen `f8` und `f9`, in denen der Status jeweils fest vordefiniert ist (siehe Listing 13.8). Mit diesen Funktionen kann man Objekte auch ein- und ausblenden.

Bei `f7` werden als Parameter die Objekt-ID und der gewünschte Status übergeben; bei den beiden anderen Funktionen reicht die Objekt-ID. Aufgerufen werden die Funktionen im `<body>`-Tag der Seite mit dem Event-Handler `onLoad` (siehe Listing 13.9).

✓ Tipp:

- Opera 5 zeigt zwar die Sichtbarkeit eines Objekts richtig an, nicht aber den Status. In Opera 6 besteht dieses Problem nicht mehr.

Problem 2

Wie wir bereits aus Kapitel 9 wissen, unterstützt Netscape 4 zwar die Werte `visible` und `hidden`, setzt sie aber in die browserspezifischen Werte `show` und `hide` um.

Beim Auslesen des Sichtbarkeitsstatus muss diese Netscape-Eigenheit berücksichtigt werden. Dazu wird nicht auf eine, sondern auf zwei Bedingungen geprüft, die durch den logischen Operator `||` (Oder) verknüpft werden (siehe Listing 13.10).

Den Status auslesen

Zum Auslesen des Status wird in der Funktion `f10` je eine `if`-Anweisung für `visible` und `hidden` geschrieben (siehe Listing 13.11). In diesen Verzweigungen wird jeweils auf zwei Bedingungen geprüft, um die Werte von Netscape 4 zu berücksichtigen. Die `else`-Anweisung tritt ein, wenn ein Browser keinen der Werte kennt.

```
(dom.visibility == 'visible') ||
(dom.visibility == 'show')
```

Listing 13.10: Besonderheiten bei Netscape 4

```
/* Den Sichtbarkeitsstatus auslesen */
function f10(ObjID) {

var dom = crossDOM(1,ObjID);

if ((dom.visibility == 'visible') ||
(dom.visibility == 'show')) {
    return 'visible';
}

if ((dom.visibility == 'hidden') ||
(dom.visibility == 'hide')) {
    return 'hidden';
}

else {
    return 'ohne';
}
}
```

Listing 13.11: Prüfung der Bedingungen für Netscape 4

```
<html><head>
<title>Sichtbarkeit</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function anzeige(ObjID) {
SichtStat = f10(ObjID);
alert("Status: " +SichtStat);
}
//--></script>
<style type="text/css"><!--
#bild1 { position:relative; top:5px;
left:0px; }
--></style></head>
<body onLoad="f8('bild1')">
<p><a href="#" onClick="anzeige('bild1')">
Status anzeigen</a><br>
<a href="#" onClick="f9('bild1')">
Objekt ausblenden</a>
<a href="#" onClick="f8('bild1')">
Objekt einblenden</a></p>
<div id="bild1"><img src="b.gif"
width="35" height="32" border="0" alt="">
</div></body></html>
```

Listing 13.12: Status am Bildschirm ausgeben

Abbildung 13.4: Anzeige des Status

Ein Beispiel für den Einsatz

In Listing 13.12 sehen Sie eine Funktion, mit der der Status am Bildschirm ausgegeben wird. Im `<body>`-Tag wird der Status dynamisch gesetzt. Die Funktionen `f8` und `f9` werden auch dazu verwendet, um das Objekt aus- und einzublenden.

1. `<script src="dom.js"></script>`
   Das CrossBrowser-DOM wird im Head der Seite referenziert.
2. `function anzeige(ObjID) {`
   Die Funktion wurde `anzeige` genannt. Als Parameter wird die Objekt-ID übergeben.
3. `SichtStat = f10(ObjID);`
   Die Variable für den Status wird erstellt: `SichtStat`. Verwenden Sie hier auf keinen Fall den Namen *status*, da das ein reserviertes Wort ist (siehe Anhang). Der Variablen wird der Rückgabewert aus Funktion `f10` zugewiesen.
4. `alert(…); }`
   Die `alert`-Anweisung gibt den Wert der Variablen in einer JavaScript-Meldung aus. Danach wird die Funktion mit der schließenden Klammer beendet.
5. `<body onLoad="f8('bild1');"`
   Im `<body>`-Tag wird der Status dynamisch gesetzt, in diesem Fall auf `visible`.

Das Ergebnis sehen Sie in Abbildung 13.4.

z-index eines Objekts

Der z-index wird über `zIndex` ermittelt. Falls Sie sich über die Schreibweise wundern: In Variablennamen sind keine Bindestriche erlaubt, daher wurde eine andere Regelung getroffen. Der Bindestrich entfällt und der folgende Buchstabe wird groß geschrieben. So wird aus z-index dann `zIndex`.

Beim Auslesen des z-index besteht das gleiche Problem wie bei der Sichtbarkeit: Einige Browser, darunter auch der IE können den z-index nur auslesen, wenn er dynamisch gesetzt wurde.

Wir benötigen also wieder eine Funktion, die beim Laden der Seite den z-index dynamisch definiert (siehe Listing 13.13). Als Parameter werden die Objekt-ID und der gewünschte z-index übergeben.

Die Funktion wird wieder bei `onLoad` der Seite aufgerufen (siehe Listing 13.14). Sie kann natürlich auch dazu verwendet werden, die Stapelreihenfolge zu ändern.

Den z-index auslesen

Die Funktion zum Auslesen des z-index ist sehr einfach aufgebaut. Der z-index wird mit der Bedingung `(dom.zIndex != null)` immer dann zurückgegeben, wenn ein Wert vorhanden ist. Die Funktion finden Sie in Listing 13.15.

```
/* Für Objekte den z-index dynamisch
setzen. */

function f11(ObjID,Stapel) {
dom = crossDOM(1,ObjID)
dom.zIndex = Stapel ;
}
```

Listing 13.13: Der z-index wird dynamisch gesetzt. Als Parameter werden die Objekt-ID und die Nummer für den gewünschten z-index übergeben.

```
<body onLoad="f11('bild1',1);
f11('bild2',2);">
```

Listing 13.14: Die Funktion wird im `<body>`-Tag mit dem Event-Handler `onLoad` aufgerufen.

```
/* Den z-index auslesen */

function f12(ObjID) {
dom = crossDOM(1,ObjID)
if (dom.zIndex != null) {
return dom.zIndex;
}
else { return (null); }
}
```

Listing 13.15: Auslesen des z-index

```
<html><head>
<title>z-index</title>
<script src="dom.js"></script>
<script language="JavaScript">
<!--
function StapAnz(ObjID) {
StapelNr = f12(ObjID);
alert("z-index: " +StapelNr);
}
//--></script>
<style type="text/css"><!--
#rot { position:absolute; top:60px;
left:40px; background:#FF0000; }
#grau { position:absolute; top:80px;
left:60px; background:#F0F0F0; }
--></style></head>
<body onLoad="f11('rot',1);
f11('grau',2);">
<p><a href="#" onClick="StapAnz('rot')">z-
index rote Ebene</a>
<a href="#"
onClick="StapAnz('grau')">z-index graue
Ebene</a><br>
<a href="#" onClick="f11('rot',1);
f11('grau',2);">rot unten</a>
<a href="#" onClick="f11('rot',2);
f11('grau',1);">rot oben</a></p>
<div id="rot"><br>Rote Ebene<br></div>
<div id="grau"><br>Graue Ebene<br></div></
body></html>
```

Listing 13.16: Anzeige des z-Index

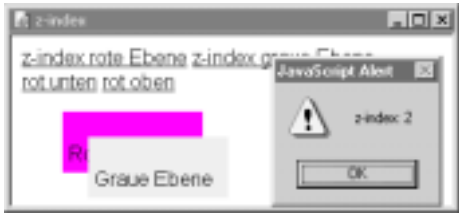

Abbildung 13.5: Browserdarstellung

Ein Beispiel für den Einsatz

In Listing 13.16 sehen Sie eine Funktion, mit der der z-index am Bildschirm ausgegeben wird. Im `<body>`-Tag wird der z-index dynamisch gesetzt. Die Funktion `f11` wird auch zur Änderung der Stapelreihenfolge verwendet.

1. `<script src="dom.js"></script>`
   Das CrossBrowser-DOM wird im Head der Seite referenziert.
2. `function StapAnz(ObjID) {`
   Die Funktion wurde `StapAnz` genannt. Als Parameter wird die Objekt-ID übergeben.
3. `StapelNr = f12(ObjID);`
   Die Variable für den Status wird erstellt: `StapelNr`. Der Variablen wird der Rückgabewert aus der Funktion `f12` zugewiesen.
4. `alert(…); }`
   Die `alert`-Anweisung gibt den Wert der Variablen in einer JavaScript-Meldung aus. Danach wird die Funktion mit der schließenden Klammer beendet.
5. `<body onLoad="f11('rot',1);"`
   Im `<body>`-Tag wir der `z-index` dynamisch gesetzt, in diesem Fall auf `1`.

Das Ergebnis sehen Sie in Abbildung 13.5.

Die Rechnerumgebung

Um die Objekteigenschaften in den letzten Abschnitten auszulesen, wurde das DOM benötigt. Zum Auslesen von Informationen, die die Rechnerumgebung betreffen, braucht man das DOM nicht. Diese Informationen werden direkt vom Besucherrechner geliefert.

In Kapitel 11 wurde die Objekthierarchie für das DOM angegeben. Dort ist `window` das oberste Objekt. Neben `window` existieren in JavaScript weitere unabhängige Objekte, dazu gehören u.a. `navigator` und `screen`.

Beide Objekte haben bestimmte Eigenschaften und können uns nützliche Informationen liefern. Mit `screen` kann man z.B. Angaben zum Bildschirm des Anwenders ermitteln. Das `navigator`-Objekt liefert Informationen über den Browser des Besuchers.

Sie finden die jeweiligen Eigenschaften in alphabetischer Reihenfolge in Tabelle 13.1 und Tabelle 13.2. In den folgenden Abschnitten werden nicht alle Eigenschaften behandelt.

Eigenschaften des Objekts navigator

EIGENSCHAFT	BEDEUTUNG
`appCodeName`	Spitzname des Browsers
`appName`	offizieller Browsername
`appVersion`	Browserversion
`cookieEnabled`	Sind Cookies erlaubt?
`language`	Sprache des Browser, z.B. Deutsch
`platform`	Plattform des Rechners
`userAgent`	Identifikation des Browsers bei HTTP-Anforderungen `p`

Tabelle 13.1: `navigator`-Eigenschaften

Eigenschaften des Objekts screen

EIGENSCHAFT	BEDEUTUNG
`availHeight`	maximal verfügbare Höhe des Browserfensters in Pixeln (im Vollbildmodus).
`availWidth`	maximal verfügbare Breite des Browserfensters in Pixeln (im Vollbildmodus).
`colorDepth`	Farbtiefe des Monitors in Bit pro Pixel.
`height`	Höhe der Bildschirmauflösung in Pixeln.
`pixelDepth`	Farbauflösung des Monitors in Bit pro Pixel.
`width`	Breite der Bildschirmauflösung in Pixeln.

Tabelle 13.2: `screen`-Eigenschaften

✓ Tipps:

- Wenn man Browserabfragen verwendet, hat das gegenüber DOM-Abfragen einen entscheidenden Nachteil: Bei der Browserabfrage muss man wissen, welche Fähigkeiten ein bestimmter Browser hat.
- Sie sollten die Browserabfrage nur dann anwenden, wenn dies durch einen bekannten Bug tatsächlich erforderlich ist.

Den Browser abfragen

Beim Aufbau des CrossBrowser-DOM haben wir festgestellt, dass man manchmal den Browsernamen und die Version benötigt, um z.B. einem Browserbug entgegenzuwirken.

Der Name und die Version eines Browsers liefern zusammen die Informationen, die man benötigt, um eine browserabhängige Funktion zu schreiben.

Browsernamen

Netscape verwendet den Browsernamen *Netscape*, beim IE ist es *Internet Explorer*. Beim Opera kann der Anwender selbst entscheiden, wie der Browser sich anmeldet: Opera, Netscape oder Internet Explorer. Die meisten anderen Browser melden sich ebenfalls als Netscape oder IE an.

Browserversionen

Die Browserversion ist nicht immer eindeutig. So meldet sich z.B. der IE5.5 als Version 4.0 an, Netscape 6.2 meldet Version 5.0, beim Opera ist die ausgegebene Version davon abhängig, welche Browseranmeldung der Benutzer eingestellt hat.

Den Browsernamen abfragen

Je nach Browser und Plattform werden verschiedene Anmelde-Strings verwendet. Irgendwo im String steht der Browsername, aber wo? Diesem Problem begegnen wir mit der Stringabfrage `indexOf`.

1. `var NN = 0;`
   Erstellen Sie im Script-Bereich vier Variablen für die Browsernamen. Als Abkürzungen wurden im Beispiel `IE` für Internet Explorer, `NN` für Netscape, `OP` für Opera und `AB` für "Anderer Browser" verwendet. Die Werte der Variablen werden auf `0` (`false`) gesetzt.
2. `if(navigator.appName.indexOf ('Netscape') != -1) { OP = 1; }`
   Der Anmelde-String wird mit `indexOf` auf den Browsernamen geprüft. Die Zählung des Index beginnt mit `0` (Null), wenn die Suche erfolglos war, wird als Wert –1 zurückgeliefert. Wenn der Wert ungleich –1 ist, wurde der Browsername gefunden. In diesem Fall setzen wir den Wert der entsprechenden Variablen auf `1` (`true`).
3. `else { AB = 1; }`
   Es wird nach den Browsernamen Netscape, Internet Explorer und Opera gesucht. Wenn keiner dieser Namen gefunden wird, handelt es sich um einen anderen Browser, der Wert für `AB` wird auf `1` (`true`) gesetzt.

Die verschachtelte Browserabfrage finden Sie in Listing.

```
var IE = 0;
var NN = 0;
var OP = 0;
var AB = 0;
if(navigator.appName.indexOf('Net
scape') != -1) {NN = 1; }
else {
if(navigator.appName.indexOf('Inter
net Explorer') != -1) { IE = 1; }
else {
if(navigator.appName.indexOf('Ope
ra') != -1) { OP = 1; }
else { AB = 1; }
}
}
```

Listing 13.17: Abfrage der Browsernamen. Die Zeilenumbrüche in den `if`-Bedingungen sind durch die Spaltenbreite im Buch bedingt.

✓ Tipp:

- Die Abfrage der Browsernamen wurde am Anfang des CrossBrowser-DOMs eingefügt. So steht auch der Browsername als globale Variable zur Verfügung.

```
browserVersion =
parseInt(navigator.appVersion);
```

Listing 13.18: Die Browserversion wird in eine Ganzzahl verwandelt.

```
var browserVersion = 0;
if(navigator.appVersion.indexOf(
'MSIE 5') != -1) {
browserVersion = 5; }
else { browserVersion =
parseInt(navigator.appVersion); }
```

Listing 13.19: Für den IE5 wird in der Abfrage die Stringsuche angewendet, um die richtige Version auszulesen.

✓ Tipp:

- Die Abfrage der Browserversion wurde am Beginn des CrossBrowser-DOMs eingefügt. So steht auch die Version als globale Variable zur Verfügung.

Die Browserversion abfragen

Wie schon erwähnt, wird die Browserversion, mit der sich ein Browser anmeldet, nicht immer zuverlässig angegeben. Der zurückgelieferte Wert von `navigator.appVersion` umfasst aber noch mehr Informationen als nur die Version.

Netscape 4.75 auf Windows ME liefert z.B. den folgenden Wert zurück:

- `4.75 [de] (Win95; U)`

IE5.5 liefert auf Windows ME folgenden Wert:

- `4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)`

Beide Browser melden die falsche Windowsversion; das spielt hier aber keine Rolle. Netscape 4 liefert die richtige Browserversion. Der IE meldet sich zwar als Version 4.0 an, in den zusätzlichen Informationen steht aber die richtige Version.

Meistens muss man nur Bugs im Netscape 4 und IE4 mit einer Browserabfrage abfangen; dazu sind die Rückgabewerte vollkommen ausreichend.

Um beim Netscape z.B. die Versionen 4.5, 4.72 und 4.75 zusammenfassen zu können, wird die Browserversion in eine Ganzzahl umgewandelt, Das geschieht mit der vordefinierten JavaScript-Funktion `parseInt` (siehe Listing 13.18).

Für den IE wenden wir zuerst die Stringabfrage an, um die tatsächliche Version herauszufinden. Für alle anderen Browser wird mit der `else`-Anweisung die Version in eine Ganzzahl verwandelt. Die komplette Abfrage der Browserversion sehen Sie in Listing 13.19.

Bildschirmabfragen

Im WWW gelten 15-Zoll-Monitore und eine Bildschirmauflösung von 800x600 immer noch als Standard. Danach sollte sich das Layout einer Webseite richten.

Wenn der Monitor des Anwenders diese Mindestvoraussetzungen nicht erfüllt, können Sie den Besucher darauf aufmerksam machen oder ihn automatisch auf Alternativseiten für niedrige Auflösungen und geringe Farbtiefe umleiten. Dazu kann man verschiedene Eigenschaften des Objekts `screen` abfragen.

Die Bildschirmauflösung abfragen

Die Auflösung, die der Besucher eingestellt hat, kann man mit `screen.height` (Höhe) und `screen.width` (Breite) ermitteln.

Das Browserfenster im Vollbildmodus

Die zur Verfügung stehende Fenstergröße im Vollbildmodus hängt vom Browser und dem Betriebssystem ab. Die meisten Browser ziehen unter Windows z.B. die Höhe (bzw. Breite, je nach Anordnung) der Windows-Taskleiste ab. Die Fensterhöhe wird mit `screen.availHeight` ermittelt, die Breite mit `screen.availWidth`.

In Listing 13.20 sind die Abfragen für die Auflösung und die maximale Fenstergröße zusammengefasst. Die Rückgabewerte der Variablen wurden mit `document.write` am Bildschirm ausgegeben (siehe die Abbildung 13.6 und Abbildung 13.7).

```
var AufloesHoehe = screen.height
var AufloesBreite = screen.width
var AnzHoehe = screen.availHeight
var AnzBreite = screen.availWidth

document.write('Die Auflösung des Moni-
tors: ' +AufloesBreite+ ' x '
+AufloesHoehe+ '<br><br>');
document.write('Im Vollbildmodus ist die
Anzeigefläche des Browsers ' +AnzBreite+
'px  breit und '
+AnzHoehe+ 'px hoch. <br><br>');
```

Listing 13.20: Abfrage der Auflösung und der Fenstergröße im Vollbildmodus.

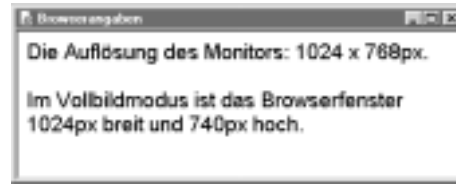

Abbildung 13.6: Die Höhe der Windows-Taskleiste wurde abgezogen, daher ist die maximale Fensterhöhe kleiner als die Höhe des Bildschirms.

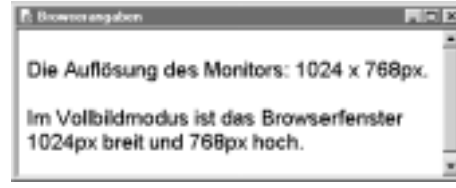

Abbildung 13.7: Hier war die Taskleiste ausgeblendet, daher entspricht die maximale Fenstergröße nun der Bildschirmauflösung.

```
function f13(BildName,res1,res2) {
dom = document.images[BildName] ;
if(screen.colorDepth > 8 ) {
return (dom.src = res2); }
else { return (dom.src = res1); }
}
```

Listing 13.21: Wenn die Farbtiefe größer als 8 Bit ist, wird das Originalbild (res2) aufgerufen, sonst das Bild mit wenigen Farben (res1).

```
<body onLoad=
"f13('bild1','c.gif','d.gif')">
<img name="bild1" src="c.gif">
```

Listing 13.22: Die Funktion wird mit dem Handler `onLoad` in das `<body>`-Tag eingebunden. Denken Sie im `<img>`-Tag an das `name`-Attribut.

Bit-Angabe	Anzahl der Farben
4	16
8	256
16	65.536
24	16,7 Millionen

Tabelle 13.3: Bit-Angaben und die Anzahl der Farben

✓ Tipp:

- Sie sollten das Herunterrechnen auf 256 Farben nicht dem Browser überlassen. Dabei kommt es zu Farbverschiebungen, die aus einem schönen Foto rasch eine stümperhaft aussehende Grafik machen. Beim Herunterrechnen in einem guten Bildbearbeitungsprogramm kann man zwar auch nicht verhindern, dass ein Foto an Qualität verliert; das Ergebnis ist aber um Einiges besser als das, was der Browser liefert.

Farbtiefe abfragen

Eine weitere Eigenschaft des Objekts `screen` ist `screen.colorDepth` für die Farbtiefe. Dieser Wert drückt aus, wie viele Farben ein Bildschirm darstellen kann. Dabei ist der angegebene Bit-Wert der Exponent zur Basis 2. Bei 16 Bit Farbtiefe kann ein Monitor also 65.536 Farben darstellen.

Früher wurden nur 256 Farben (8 Bit) unterstützt. Diese Bildschirme werden zwar immer seltener verwendet, es gibt aber immer noch etliche Anwender, die mit solchen Computern durchs Netz surfen. Darauf sollten Sie nach Möglichkeit bei der Erstellung von Webgrafiken Rücksicht nehmen. Das GIF-Format ist auf 256 Farben begrenzt, nicht aber das JPG-Format, das normalerweise für Fotos verwendet wird.

Es macht natürlich wenig Sinn, grundsätzlich alle Fotografien auf 256 Farben herunterzurechnen, denn dabei gehen oft zu viele Bildinformationen verloren. Durch die Abfrage der Farbtiefe kann man aber auf ältere Monitoren und Grafikkarten reagieren. Legen Sie Kopien der Fotos an und rechnen Sie diese auf 256 Farben herunter. Die Kopien werden nur dann aufgerufen, wenn der Besucherrechner das Foto in den Originalfarben nicht anzeigen kann.

In Listing 13.21 finden Sie ein Beispiel für den praktischen Einsatz. Wie viele Farben ein Monitor bei den verschiedenen Bit-Angaben darstellen kann, sehen Sie in Tabelle 13.3.

Fensterabfragen

Neben `navigator` und `screen` kann auch das Objekt `window` einige nützliche Informationen liefern.

Die Größe des geöffneten Browserfenster

Über das `screen`-Objekt kann man herausfinden, wie groß das Browserfenster im Vollbildmodus sein kann. Für ein Fenster, das nicht im Vollbildmodus geöffnet ist, kann man die äußeren Dimensionen mit `window.outerHeight` und `window.outerWidth` feststellen.

Dabei werden auch die Programmtitelleiste, die Menüleiste, Symbolleisten und die Statusleiste berücksichtigt.

Der Internet Explorer unterstützt diese beiden Eigenschaften nicht, im Opera kann man die Werte auslesen, aber nicht ändern. Netscape kann die Werte lesen und ändern.

In Listing 13.23 sehen Sie ein Beispiel, die Abbildung 13.8 und Abbildung 13.9 zeigen das Ergebnis im Netscape 4. Die Fenstergröße wurde geändert, damit die Anzeige der Werte aktualisiert wird, muss die Seite neu geladen werden.

```
<html><head>
<title>Fensterabfragen</title>
<style type="text/css"><!--
p,body,a { font:14pt Arial; }
-->
</style></head>
<body>
<script language="JavaScript">
<!--
document.write('Das Fenster ist '
+window.outerWidth+ 'px breit und');
document.write(window.outerHeight+
'px hoch.<br>');
//--></script>
<a href="#" onClick="javascript:
window.outerWi
dth=250">Fensterbreite auf 250px setzen
</a></body></html>
```

Listing 13.23: Die Größe des Browserfensters kann in Netscape gelesen und geändert werden.

Abbildung 13.8: Die Seite nach dem Laden

Abbildung 13.9: Das Fenster wurde höher.

```
function f14() {
if(window.innerHeight != null) {
return (window.innerHeight); }
if(document.body.clientHeight != null) {
return (document.body.clientHeight); }
else { return (null); }
}


function f15() {
if(window.innerWidth != null) {
return (window.innerWidth); }
if(document.body.clientWidth != null) {
return (document.body.clientWidth); }
else { return (null); }
}
```

Listing 13.24: Die Größe der Anzeigefläche feststellen

```
function anzeigen() {
hoehe = f14();
breite = f15();
alert('Das Fenster ist ' +hoehe+ 'px hoch
und ' +breite+ 'px breit.'); }
```

Listing 13.25: Hier werden die Werte aus den Funktionen `f14` und `f15` verwendet, um die Größe der Anzeigefläche in einer `alert`-Meldung auszugeben.

Die Größe der Anzeigefläche

Interessanter als die äußeren Dimensionen des Browserfensters sind die Abmessungen der Anzeigefläche. Damit ist der Bereich gemeint, in dem die Webseite angezeigt wird, ohne Symbolleisten oder sonstige Bedienelemente.

Um die Größe der Anzeigefläche auszulesen, werden die Eigenschaften `window.innerHeight` und `window.innerWidth` für Netscape verwendet, der Internet Explorer benutzt dazu die Eigenschaften `document.body.clientHeight` und `document.body.clientWidth`.

Wir benötigen zwei Funktionen: eine zum Auslesen der Breite und eine für die Höhe. In beiden Funktionen wird jeweils geprüft, ob die Eigenschaften ungleich null sind. Falls das der Fall ist, werden als Werte die Abmessungen in Pixeln zurückgeliefert.

Sie finden die Funktionen in Listing 13.24. Die Rückgabewerte können in anderen Funktionen verwendet werden (siehe Listing 13.25).

Resize-Bug abfangen

Netscape 4 hat einen schwerwiegenden CSS-Bug. Immer, wenn eine externe CSS-Datei verwendet wird und der Besucher die Größe des Browserfensters ändert, verschwinden die CSS-Stile wie von Zauberhand. Erst wenn die Seite neu geladen wird, werden die Stile wieder angezeigt.

Dieser Bug sollte unbedingt abgefangen werden. Im schlimmsten Fall zeigt Netscape 4 überhaupt nichts mehr an, wenn alle Objekte mithilfe von IDs erzeugt wurden (siehe die Abbildung 13.10 und Abbildung 13.11).

Man muss Netscape 4 also dazu bringen, die Seite neu zu laden, wenn der Besucher die Größe des Browserfensters ändert.

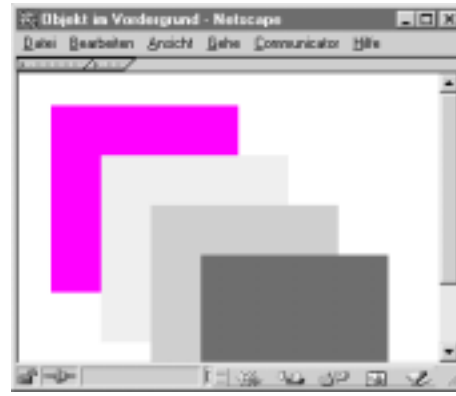

Abbildung 13.10: Die Seite nach dem Laden. Die Objekte sind nicht vollständig sichtbar, also vergrößert der Besucher das Browserfenster…

Abbildung 13.11: …und sieht dann überhaupt nichts mehr, da alle Seitenelemente durch Objekt-IDs in einer externen CSS-Datei erzeugt werden.

```
if (document.layers) {
LadeBreite = innerWidth;
LadeHoehe = innerHeight;
}

function f16() {
if (innerWidth != LadeBreite
|| innerHeight != LadeHoehe)
location.reload();
}
if(document.layers) onResize = f16;
```

Listing 13.26: Den Resize-Bug in Netscape 4 durch Neuladen der Seite abfangen.

✓ Tipps:

- Das Objekt `location` ist wie `document` ein Unterobjekt von `window`. Über location hat man Zugriff auf die URL, also die Adresse, die über eine Datei im Internet verfügt. Die Methode `reload()` hat die gleiche Wirkung wie der *Reload*-Button im Browser: Die Webseite wird neu geladen.
- Sie sollten das Script auf allen Seiten verwenden, in denen CSS extern benutzt werden. Wenn Sie in einer Webseite kein DHTML, sondern nur CSS verwenden, benötigen Sie die anderen Funktionen der Datei *dom.js* nicht. Speichern Sie das Script dann alleine in einer externen Datei, z.B. *neuladen.js*, und binden Sie statt der *dom.js* diese Datei ein. So spart man etwas Ladezeit.

Die Funktion zum Neuladen

1. Beim Laden der Seite wird die Größe der Anzeigefläche in den globalen Variablen `LadeBreite` und `LadeHoehe` gespeichert, wenn der Browser das Layer-DOM verwendet.
2. Die Funktion `f16` vergleicht die aktuelle Breite und Höhe der Anzeigefläche mit den Werten der beiden Variablen `LadeBreite` und `LadeHoehe`. Wenn sie nicht übereinstimmen, wird die Methode `location.reload();` aufgerufen.
3. Der Event-Handler `onResize` reagiert, wenn die Größe des Browserfensters geändert wird. Die Funktion `f16` wird immer dann aufgerufen, wenn der Browser das Layer-DOM verwendet und die Größe des Browserfensters geändert wird.

Das Script im Einsatz

Die Funktion wurde in der Datei *dom.js*, die in die Webseiten eingebunden wird, gespeichert.

Wenn ein Besucher nun mit Netscape 4 eine Seite aufruft, in der eine externe CSS-Datei eingebunden ist, wird die Seite automatisch neu geladen, wenn die Größe des Browserfensters geändert wird (siehe Abbildung 13.12 und Abbildung 13.13).

So sieht der Besucher auch in Netscape 4 weiterhin das, was er sehen soll: die CSS-Stile, die Sie für die Seite festgelegt haben.

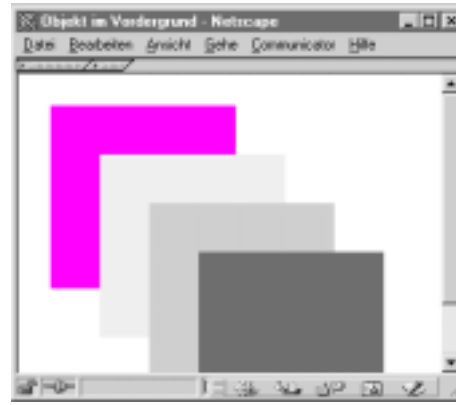

Abbildung 13.12: Die Seite nach dem Laden

Abbildung 13.13: Auch nachdem die Größe des Browserfensters geändert wurde, werden die CSS-Objekte angezeigt, da die Seite automatisch neu geladen wurde.

14

EINFACHE DHTML-TECHNIKEN

Einige einfache DHTML-Effekte haben Sie im letzten Kapitel bereits kennengelernt, z.B. das Ein- und Ausblenden von Ebenen.

In diesem Kapitel beschäftigen wir uns näher mit einfachen DHTML-Techniken. Für einige Effekte benötigt man nur die bereits vorhandenen Funktionen im *dom.js*, für andere Effekte werden weitere Funktionen hinzugefügt. Auf diese Bibliothek können Sie immer wieder zurückgreifen.

Allgemeines

Für die Beispiele in diesem Kapitel wurde die Datei *dom.js* erweitert, sofern neue Scripts benötigt werden. Die Funktionen für die Basiseffekte werden mit `e` benannt (e für Effekte) und ebenfalls durchnumeriert.

Wenn Sie häufig mit dem CrossBrowser-DOM arbeiten, werden Sie die Nummern der Funktionen schnell im Kopf behalten können. Für den Anfang sollten Sie aber die Funktionsübersicht in eine Textdatei kopieren, ausdrucken und neben die Tastatur legen.

In Listing 14.1 sehen Sie die Übersicht der bisherigen Funktionen des CrossBrowser-DOMs noch einmal in Kurzform. In der Datei *dom.js* ist zusätzlich vermerkt, welche Parameter übergeben werden müssen.

```
/*Übersicht der Funktionen:

f1: Höhe eines Objekts feststellen

f2: Breite eines Objekts feststellen

f3: Position top

f4: Position left

f5: Position unten, Werte aus f1 und f3

f6: Position rechts, Werte aus f2 und f4

f7: Sichtbarkeitsstatus visible
    oder hidden setzen

f8: Nur Sichtbarkeit visible setzen

f9: Nur Sichtbarkeit hidden setzen

f10: Den Sichtbarkeitsstatus auslesen

f11: z-index dynamisch setzen

f12: Den z-index auslesen

f13: Farbtiefe auslesen

f14: Höhe der Browser-Anzeigefläche

f15: Breite der Anzeigefläche

f16: Resize-Bug in N4 abfangen

Ende der Übersicht */
```

Listing 14.1: Die Übersicht der Funktionen aus *dom.js* in Kurzform.

```
<html><head>
<title>Bilderschau</title>
<script src="dom.js"></script>
<style type="text/css">
<!--
#bild1 { position:absolute;
visibility:hidden; top:70px;
left:20px; }
#bild2 { position:absolute;
visibility:hidden; top:70px;
left:20px; }
#bild3 { position:absolute;
visibility:hidden; top:70px;
left:20px; }
#text { position:absolute; top:70px;
left:20px; width:380px; padding:5px;
}
a { font:bold 12pt Arial,sans-serif;
text-decoration:none; color:#606060;
}
body { margin-left:10px; }
p { font:11pt Arial,sans-serif;
text-align:justify; color:#606060; }
-->
</style></head>
```

Listing 14.2: Der Head-Bereich der Seite für die Bildershow

✓ Tipp:

- Da die Fotos den Text vollständig überdecken sollen, muss die Breite der Textebene begrenzt werden. Die Fotos sind 400px breit; die Breite der Textebene wurde entsprechend auf 380px begrenzt. Sie können die Textebene aber auch jeweils ein- oder ausblenden, dann ist die Begrenzung der Breite nicht erforderlich. Das ist auf jeden Fall dann erforderlich, wenn Sie einen längeren Text verwenden.

Objekte ein- und ausblenden

Das Ein- und Ausblenden von Objekten haben Sie bereits kennen gelernt (siehe die Funktionen `f7` bis `f9` in Kapitel 13). Da das Ein- und Ausblenden die Grundlage für viele dynamische Effekte bildet, wird dieses Thema hier noch einmal ausführlicher behandelt.

Als Beispiel verwenden wir eine Bildershow mit drei Fotos und einem kurzen Begrüßungstext. Für den Text und jedes Bild wird eine Objekt-ID angelegt.

Die Objekt-IDs anlegen:

1. `#bild1 { position:absolute; visibility: hidden; top:70px; left:20px; }`
   Legen Sie eine Objekt-ID für das erste Bild an und geben Sie die Definitionen ein. Die Ebenen werden absolut positioniert, die Sichtbarkeit wird auf `hidden` gesetzt.
2. Wiederholen Sie Schritt 1 für die Objekt-IDs der beiden anderen Bilder. Der Einfachheit halber kopiert man die Regel für `bild1` und ändert die Namen der IDs.
3. `#text { position:absolute; top:60px; left:20px; width:380px; padding:5px; }`
   Auch für den Text wird eine ID angelegt, die Position ist mit denen der Bilder identisch.

In Listing 14.2 sehen Sie den Head-Bereich der Webseite. Für die Tags `<a>`, `<body>` und `<p>` wurden ebenfalls Stilregeln angelegt.

Funktionen aufrufen

Zum Setzen der Sichtbarkeit für die Ebenen werden die Funktionen `f8` (sichtbar) und `f9` (nicht sichtbar) verwendet. Sie werden in den Links mit dem Event-Handler `onClick` aufgerufen. Als Parameter wird die jeweilige Objekt-ID übergeben.

1. `<a href="#" onClick="f8('bild1');f9('bild2'); f9('bild3');">Bild 1</a>`
   Der Link für Bild 1 setzt die Sichtbarkeit für die ID `bild1` auf `visible`, für `bild2` und `bild3` auf `hidden`.
2. `<a href="#" onClick="f9('bild1');f8('bild2'); f9('bild3');">Bild 2</a>`
   Der Link für Bild 2 setzt die Sichtbarkeit für die ID `bild2` auf `visible`, für `bild1` und `bild3` auf `hidden`.
3. `<a href="#" onClick="f9('bild1');f9('bild2'); f8('bild3');">Bild 3</a>`
   Der Link für Bild 3 setzt die Sichtbarkeit für die ID `bild3` auf `visible`, für `bild1` und `bild2` auf `hidden`.
4. `<a href="#" onClick="f9('bild1');f9('bild2'); f9('bild3');">Keine Bilder</a>`
   Der Link "Keine Bilder" setzt die Sichtbarkeit für `bild1`, `bild2` und `bild3` auf `hidden`. Der Text wird wieder sichtbar.

In Listing 14.3 sehen Sie den Body der Seite mit den entsprechenden Ebenen.

```
<body><p class="menutext">
<a href="#"
onClick="f8('bild1');f9('bild2');
f9('bild3');">Bild 1</a> |
<a href="#"
onClick="f9('bild1');f8('bild2');
f9('bild3');">Bild 2</a> |
<a href="#"
onClick="f9('bild1');f9('bild2');
f8('bild3');">Bild 3</a> |
<a href="#"
onClick="f9('bild1');f9('bild2');
f9('bild3');">Keine Bilder</a>
<hr noshade width="410" align="left"
size="1"></p>
<div id="text"><p>Auf dieser Seite können
Sie sich einige Fotos ansehen, die während
des letzten Urlaubs entstanden sind. <br>
Wir sind mit dem Motorrad kreuz und quer
über Mallorca gefahren und konnten dabei
Eindrücke der wunderschönen Insel einfan-
gen.
[ …mehr Text… ]</p></div>
<div id="bild1"><img src="bilder/
bild01.jpg" width="400" height="269"
border="0" alt=""></div>
<div id="bild2"><img src="bilder/
bild02.jpg" width="400" height="269"
border="0" alt=""></div>
<div id="bild3"><img src="bilder/
bild03.jpg" width="400" height="269"
border="0" alt=""></div>
</body></html>
```

Listing 14.3: Der Body der Seite für die Bildershow.

Abbildung 14.1: Die Seite nach dem Laden. Wenn man den Link Keine Bilder anklickt, sieht die Seite genauso aus.

Abbildung 14.2: Die Seite, nachdem der Link Bild 2 angeklickt wurde.

Die Seite im Browser

Listing 14.2 und Listing 14.3 ergeben zusammen den Quelltext für die Seite. Da für die IDs `bild1`, `bild2` und `bild3` die Sichtbarkeit auf `hidden` gesetzt wurde, erscheint beim Laden der Seite nur der Text. Der Link *Keine Bilder* setzt ebenfalls den Status `hidden` für alle Bildebenen; daher sieht die Seite nach dem Anklicken dieses Links genauso aus wie nach dem Laden (siehe Abbildung 14.1).

Wenn einer der Links angeklickt wird, ruft der Event-Handler `onClick` die jeweiligen Funktionen auf. In Abbildung 14.2 wurde der Link für *Bild 2* angeklickt. Die Ebene `bild2` wird eingeblendet und verdeckt den Text. Falls eine der anderen Bildebenen bereits eingeblendet war, wird diese Ebene ausgeblendet.

Wenn Sie auf der Seite einen längeren Text verwenden, muss auch die Textebene mit `f9('text')` ausgeblendet werden.

Ein Objekt um einen bestimmten Wert verschieben

Wie man für ein Objekt einen neuen Startpunkt festlegt, wurde bereits im letzten Kapitel behandelt. Dabei wurde die neue Pixelposition angegeben.

Häufig möchte man Objekte aber um einen bestimmten Wert verschieben. Eine solche Verschiebung kann mehrfach ausgeführt werden (siehe Abbildung 14.3).

Netscape 4 stellt zum Verschieben der Objekte die Methode `moveBy()` zur Verfügung; die anderen Browser können damit aber nichts anfangen. Wir benötigen also ein neues Script, das in allen gängigen Browsern funktioniert. Es wird der Datei *dom.js* hinzugefügt.

Die neue Funktion ermittelt die aktuellen Werte für die Positionen `top` und `left`, addiert die Werte für die Verschiebung und setzt die Startposition entsprechend neu.

Abbildung 14.3: Die Grafik wurde als Link angelegt. Der Event-Handler `onClick` verschiebt die Grafik bei jedem Mausklick um jeweils 70px nach rechts und 40px nach unten.

```
function e1(ObjID,umH,umV) {
/* 2 Variablen erforderlich: mit und ohne
style */

var dom = crossDOM(0,ObjID);
var domStil = crossDOM(1,ObjID);

if (domStil.pixelTop) {
domStil.pixelTop += umV;
domStil.pixelLeft += umH;
}
else {
if (dom.offsetTop) {
var oben = dom.offsetTop;
var links = dom.offsetLeft;
domStil.top = umV + oben;
domStil.left = umH + links;
}
else dom.moveBy(umH,umV);
}
}
```

Listing 14.4: Die Funktion zum Verschieben eines Objekts um bestimmte Werte

Die Funktion zum Verschieben

Die neue Funktion wird `e1` genannt. Als Parameter werden die Objekt-ID sowie die Werte übergeben, um die das Objekt verschoben werden soll. Der Parameter `umH` bezeichnet dabei den Wert für die horizontale, `umV` den Wert für die vertikale Verschiebung.

Je nach Browser und verwendetem DOM werden wieder verschiedene Anweisungen benötigt, wie auch bereits bei den Funktionen `f3` und `f4`.

1. Bei den Eigenschaften `pixelLeft` und `pixelTop` kann man den Wert mit dem Operator += direkt um die Verschiebungswerte `umH` bzw. `umV` erhöhen.
2. Einige Browser (z.B. IE und N6) verwenden zum Auslesen der aktuellen Startpositionen die Eigenschaften `offsetTop` und `offsetLeft`. Die `offset`-Eigenschaften können nur gelesen, nicht geändert werden. Bei diesen Browsern kann man aber mit `domStil.top` bzw. `domStil.left` die Startposition dynamisch neu definieren. Dazu werden die Werte der aktuellen Position den Variablen `oben` und `links` zugewiesen. Zu diesen Werten werden dann die entsprechenden Werte der Verschiebung addiert und so die neuen Startpositionen gesetzt.
3. Für Netscape 4 nutzen wir die Methode `moveBy`. Die Reihenfolge der Parameter ist für diese Methode vordefiniert: zuerst der Wert, um den das Objekt horizontal verschoben wird, dann der vertikale Wert.

In Listing 14.4 sehen Sie die neue Funktion, die der Datei *dom.js* hinzugefügt wurde.

Einsatz der Funktion

1. `<script src="dom.js"></script>`
   Im Head-Bereich der Seite wird die Datei *dom.js* eingebunden, in der sich auch die Funktion `el` zum Verschieben befindet.
2. `#bild1 { … Definitionen … }`
   Die Objekt-ID mit den gewünschten Definitionen wird ebenfalls im Head notiert.
3. `<a href="#" onClick="el('bild1',20,0)">`
   Im Seitenkörper wird mit einem Event-Handler die Funktion aufgerufen. Dabei werden die Objekt-ID und die Werte der Verschiebung als Parameter übergeben.

Wenn das Objekt nur in eine Richtung verschoben werden soll, wird für die andere Richtung der Wert `0` (`Null`) übergeben. Im Beispiel wird das Objekt mit jedem Mausklick um 20 Pixel nach rechts verschoben, an der vertikalen Position ändert sich nichts.

In Listing 14.5 sehen Sie den Quelltext der Seite. Die Abbildung 14.4 und Abbildung 14.5 zeigen das Ergebnis im Browser. Jedes Mal, wenn man den Link angeklickt, wird die Grafik 20 Pixel nach rechts verschoben. Um den Effekt zu verdeutlichen, wird der Rahmen der Grafik angezeigt und als Hintergrundbild ein Raster verwendet, dessen Linien einen Abstand von 20px haben.

```
<html><head>
<title>Objekte verschieben</title>
<script src="dom.js"></script>
<style type="text/css">
#bild1 { position:absolute; top:80px;
left:40px; margin:0px; padding:0px;}
body { background:url(bg.gif) #FFFFFF;
margin:20px; }
</style></head>
<body><p>Abstand der Rasterlinien:
20 Pixel <br>
<a onClick="el('bild1',20,0)"
href="#">Bild um 20px nach rechts
verschieben</a></p>
<div id="bild1"><img src="b.gif"
width="35" height="32" border="1">
</div></body></html>
```

Listing 14.5: Ein Objekt verschieben

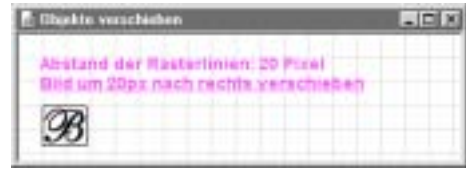

Abbildung 14.4: Die Seite nach dem Laden

Abbildung 14.5: Die Seite, nachdem der Link dreimal angeklickt wurde

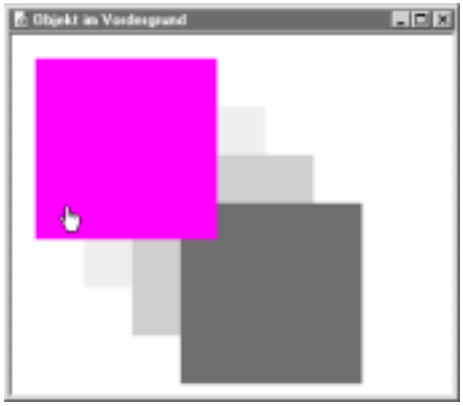

Abbildung 14.6: Die rote Ebene wurde anklickt, und dadurch nach vorne geholt.

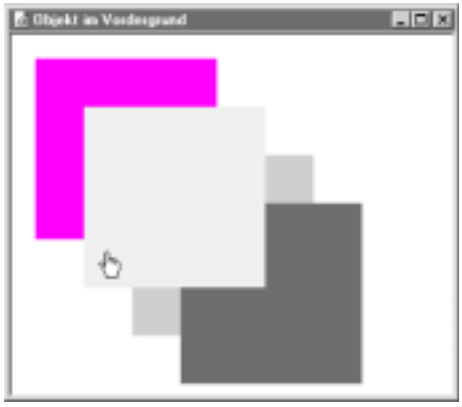

Abbildung 14.7: Dann wurde die hellgraue Ebene durch einen Klick nach vorne geholt. Die rote Ebene erhält wieder ihre ursprüngliche Stapelreihenfolge.

Objekte in den Vordergrund holen

Wie man die Reihenfolge für zwei Ebenen tauscht, wurde bereits im letzten Kapitel gezeigt. Mit mehrfachem Aufruf der Funktion `f11` wurde der z-index für jede Ebene gesetzt. Das bringt ein Objekt zwar auch in den Vordergrund, wenn aber mehr als zwei Objekte vorhanden sind, erfordert dieses Vorgehen jedoch viel Schreibarbeit und wird rasch unübersichtlich.

Sinnvoller ist eine Funktion, die ein Objekt in den Vordergrund holt und für das vorher ausgewählte Objekt wieder den ursprünglichen z-index setzt.

Die Funktion für den z-index

Die neue Funktion wurde `e2` genannt; sie greift auf die Funktionen `f11` (z-index setzen) und `f12` (z-index lesen) zurück. Damit das Script funktioniert, muss im `<body>`-Tag mit dem Event-Handler `onload` der z-index für jedes Objekt dynamisch gesetzt werden (siehe Kapitel 13). Außerdem müssen zwei globale Variablen definiert werden, die sich auf das zuletzt ausgewählte Objekt beziehen.

1. `var ObjektVorher = null;`
   Außerhalb der neuen Funktion wird die globale Variable `ObjektVorher` definiert, der Wert wird auf `null` gesetzt. In dieser Variablen wird die Objekt-ID eines vorher gewählten Objekts gespeichert.
2. `var StapelVorher = 0;`
   Die globale Variable `StapelVorher` wird auf `0` (`false`) gesetzt. In dieser Variablen wird der z-index eines vorher gewählten Objekts gespeichert.
3. `function e2(ObjID) { Anweisungen }`
   Nach den beiden globalen Variablen wird die neue Funktion mit den Anweisungen definiert.

Sie finden die Funktion in Listing 14.6. Die Wirkungsweise des Scripts wird auf der nächsten Seite erläutert. Dort steht auch der Quelltext für die Beispielseite, die Sie in Abbildung 14.6 und Abbildung 14.7 sehen konnten. Die Ebenen für das Beispiel wurden in der Datei *ebenen.css* definiert (siehe Listing 14.7).

```
/*Die Variablen ObjektVorher und Stapel-
Vorher müssen als globale Variablen defi-
niert sein!*/

var ObjektVorher = null;
var StapelVorher = 0;

function e2(ObjID) {

if(ObjektVorher != null) {
f11(ObjektVorher,StapelVorher);
}

StapelVorher = f12(ObjID);
ObjektVorher = ObjID;
f11(ObjID,1000);

}
```

Listing 14.6: Definition der globalen Variablen und die Funktion, die ein Objekt in den Vordergrund holt

```
#rot { position:absolute; top:20px;
left:20px; border:none; back-
ground:#FF0000; z-index:0; }
#hgrau { position:absolute; top:60px;
left:60px; border:none; back-
ground:#F0F0F0; z-index:1; }
#mgrau { position:absolute; top:100px;
left:100px; border:none; back-
ground:#D0D0D0; z-index:2; }
#dgrau { position:absolute; top:140px;
left:140px; border:none; back-
ground:#707070; color:#FFFFFF; z-index:3;
}
a { text-decoration:none; }
```

Listing 14.7: Der Inhalt der Datei *ebenen.css*, in der die Objekte definiert wurden

```
<html><head>
<title>Objekt im Vordergrund</title>
<link rel="stylesheet" href="ebenen.css"
type="text/css">
<script src="dom.js"></script>
</head>
<body onLoad="f11('rot',0);
f11('hgrau',1);f11('mgrau',2);
f11('dgrau',3);">
<div id="rot"><a href="#"
onClick="e2('rot');"><img src="blind.gif"
width="150" height="150" border="0"
alt=""></a></div>
<div id="hgrau"><a href="#"
onClick="e2('hgrau')"><img src="blind.gif"
width="150" height="150" border="0"
alt=""></a></div>
<div id="mgrau"><a href="#"
onClick="e2('mgrau')"><img src="blind.gif"
width="150" height="150" border="0"
alt=""></a></div>
<div id="dgrau"><a href="#"
onClick="e2('dgrau')"><img src="blind.gif"
width="150" height="150" border="0"
alt=""></a></div>
</body></html>
```

Listing 14.8: Der Quelltext der Beispielseite. Da die Objekte bereits vorhanden sein müssen, damit sie im `<body>` mit `onLoad` angesprochen werden können, muss das Script *dom.js* nach der CSS-Datei eingebunden werden. Das Ergebnis im Browser haben Sie bereits zu Beginn des Abschnitts gesehen (Abbildung 14.6 und Abbildung 14.7).

So funktioniert das Script:

Um zu verstehen, wie das Script funktioniert, muss man es quasi rückwärts lesen.

1. Wenn man nach dem Laden der Seite ein Objekt anklickt, wird mit der Funktion `f12` zuerst der z-index dieses Objekts ausgelesen und in `StapelVorher` gespeichert. In `ObjektVorher` wird die Objekt-ID dieses Elements gespeichert.
2. Anschließend wird mithilfe der Funktion `f11` der z-index des angeklickten Objekts auf `1000` gesetzt. Mit diesem hohen z-index ist auch bei vielen Elementen auf der Seite sichergestellt, dass das Objekt in der Stapelreihenfolge wirklich oben liegt.
3. Wenn man das nächste Objekt anklickt, wird durch `if(ObjektVorher != null)` festgestellt, dass `ObjektVorher` nicht mehr leer ist, also wird die Funktion `f11` mit den Parametern `StapelVorher` und `ObjektVorher` aufgerufen. Durch die darin gespeicherten Werte wird für das vorher angeklickte Objekt der z-index wieder auf den ursprünglichen Wert gesetzt und das Objekt somit wieder in seine alte Stapelreihenfolge gebracht.
4. In `StapelVorher` und `ObjektVorher` werden nun der z-index und die Objekt-ID des neu angeklickten Objekts gespeichert, dann der z-index auf `1000` gesetzt usw.

In Listing 14.8 finden Sie den Quelltext der Beispielseite. Das Ergebnis im Browser haben Sie bereits in Abbildung 14.6 und Abbildung 14.7 zu Beginn dieses Abschnitts gesehen. Beachten Sie, dass die Datei *ebenen.css* vor dem Script *dom.js* referenziert werden muss!

Grafiken vorausladen

Oft werden auf Webseiten bei Navigationsleisten Grafiken mit Rollover-Effekten verwendet. Beim Überfahren mit der Maus wird die Button-Grafik gegen eine andere ausgetauscht (siehe auch den nächsten Abschnitt *Rollover-Effekte*).

Grafiken benötigen Ladezeit und werden dadurch normalerweise mit Verzögerung angezeigt. Wenn die Grafiken vorgeladen werden, stehen sie im Cache des Browsers bereits zur Verfügung und die Anzeige erfolgt dann ohne Verzögerung.

Die Funktion zum Vorausladen

1. `var grafikLaden = new Array();`
   Zum Laden der Bilder wird eine neue Instanz des `Array`-Objekts erzeugt. Die Variable wird global erstellt.
2. In der Funktion `e3` wird die Variable `Bild` erstellt, der mit `e3.arguments` der Wert des jeweils übergebenen Parameters zugewiesen wird. In unserem Fall sind die Parameter, die der Funktion in der Webseite übergeben werden, die Pfadangaben der Grafikdateien.

Fortsetzung auf der nächsten Seite

```
var grafikLaden = new Array();

function e3() {
if(document.images) {
var Bild = e3.arguments;
for (var i=0; i <
e3.arguments.length; i++) {
    grafikLaden[i] = new Image;
    grafikLaden[i].src = Bild[i];
    }
}
}
```

Listing 14.9: Die Funktion zum Vorausladen von Grafiken

```
<body onLoad="e3('bild.gif','bilder/
bild17.jpg','bilder/thumbs/klein3.jpg')">
```

Listing 14.10: Die Funktion wird mit `onLoad` im `<body>`-Tag der Seite aufgerufen.

3. Die Anzahl der Grafikdateien, die der Funktion beim Aufruf in der Webseite übergeben werden, wird mit `e3.arguments.length` ermittelt. Mit `e3.arguments[0]` wird der Wert des ersten übergebenen Parameters angesprochen, mit `e3.arguments[1]` der des zweiten Parameters usw.
4. Wir verwenden zum Ansprechen der Parameterwerte eine `for`-Schleife, in der der Zähler `i` bei jedem Durchlauf solange erhöht wird, bis alle übergebenen Parameter abgearbeitet wurden.
5. Für jede Grafikdatei des Arrays `grafikLaden` wird mit dem reservierten Wort `new` und der JavaScript-Objektfunktion `Image()` eine neue Instanz erzeugt.
6. Der neuen Instanz wird dann mit der Objekteigenschaft `src` die entsprechende Grafikdatei zugeordnet.

Sie finden die Funktion in Listing 14.9. Im `<body>`-Tag der Webseite wird mit dem Handler `onLoad` die Funktion aufgerufen. Als Parameter werden dabei die Grafikdateien angegeben, die geladen werden sollen. Wenn die Grafiken in einem anderen Verzeichnis liegen als die Webseite, muss der relative Pfad mit angegeben werden (siehe Listing 14.10).

✓ Tipps:

- Das Vorausladen von Grafiken bietet sich auch dann an, wenn Sie auf einer nachfolgenden Seite ein größeres Foto anzeigen wollen.
- Die Ladezeit der Seite verlängert sich durch das Vorausladen von Grafiken. Vermeiden Sie daher, mehrere große Grafikdateien zusammen vorauszuladen.

Rollover-Effekte

Navigationsbuttons mit Rollover-Effekten werden häufig auf Webseiten verwendet. Beim Überfahren mit der Maus wird dabei eine andere Grafik angezeigt. Wenn der Mauszeiger den Link verlässt, wird die Grafik wieder ausgetauscht. Dieser Effekt wird häufig auch *Hover-Effekt* genannt.

Um den Effekt zu erzeugen, kann man in einem Link für die Event-Handler mit der Eigenschaft `src` direkt eine neue Grafikdatei festlegen (siehe Listing 14.11). Dabei muss man aber jedes Mal `document.images['name'].src='grafik.gif'` notieren.

Mit einer einfachen Funktion spart man sich einige Schreibarbeit. Der Name der Grafik und die Grafikdatei werden dabei als Parameter übergeben, das erneute Eingeben von `document.images['name'].src` entfällt dadurch.

```
<a href="#" onmouseover= "docu-
ment.images['button3'].src=
'button03_over.gif';"
onmouseout= "docu-
ment.images['button3'].src=
'button03.gif';">
<img name="button3" src="button03.gif"
width=120 height=30 border=0></a>
```

Listing 14.11: Bei der direkte Eingabe muss jedes Mal der komplette DOM-Pfad der Grafik notiert werden.

```
<a href="#" onmouseover=
"e4('button3','button03_over.gif');"
onmouseout=
"e4('button3','button03.gif');">
<img name="button3" src="button03.gif"
width=120 height=30 border=0></a>
```

Listing 14.12: Durch die einfache Funktion spart man einige Schreibarbeit.

```
function e4() {
if (document.images) {
document[e4.arguments[0]].src=
e4.arguments[1];
}
}
```

Listing 14.13: Die Funktion zum Austauschen der Grafiken

Abbildung 14.8: Die Webseite nach dem Laden

Abbildung 14.9: Beim Überfahren des Buttons wird eine andere Grafik angezeigt. Wenn der Mauszeiger den Linkbereich verlässt, wird wieder die ursprüngliche Grafik eingeblendet.

Die Rollover-Funktion

Beim Aufruf in der Webseite wird als erster Parameter der Name der Grafik im `<img>`-Tag übergeben. Der zweite Parameter ist die Grafikdatei, die für den Event-Handler verwendet werden soll (siehe Listing 14.12).

1. Für `document.images` wird mit `e4.arguments[0]` der Wert des ersten übergebenen Parameters angesprochen, also der Name der Grafik im `<img>`-Tag.
2. `document[e4.arguments[0]].src` spricht die Eigenschaft `src` des Grafikobjekts an.
3. `e4.arguments[1]` weist der Eigenschaft `src` den Wert des zweiten Parameters zu, also die angegebene Grafikdatei.

Wenn der Aufruf der Funktion z.B. `e4('button3','button03.gif')` lautet, übersetzt der Browser die Funktion mit `document.images['button3'].src= 'button03.gif'`.

In Abbildung 14.8 und Abbildung 14.9 sehen Sie ein Beispiel für den Rollover-Effekt.

✓ Tipp:

- Für den Rollover-Effekt eignet sich das Vorausladen der Grafiken, die beim Event-Handler `onmouseover` angezeigt werden sollen.
- Durch das Hinzufügen des Event-Handlers `onMouseDown` kann beim Anklicken des Buttons eine weitere Grafik angezeigt werden.
- Optimieren Sie die Button-Grafiken so weit wie möglich. Je kleiner die Grafiken für den Rollover-Effekt, desto besser.

Ein neues Browserfenster öffnen

Popup-Fenster findet man im WWW häufig. Manche Provider verwenden sie zur Einblendung von Werbebannern; sie können aber auch anderen Zwecken dienen. Man kann daran z.B. zusätzliche Informationen anzeigen oder eine Fernsteuerung für die Webseite realisieren.

JavaScript stellt zum Öffnen eines neuen Fensters für das Objekt `window` die Methode `open()` zur Verfügung. Es müssen mindestens zwei Parameter übergeben werden: die Datei, die im Fenster geöffnet werden soll und der Fenstername.

Außerdem kann das Aussehen des neuen Fensters bestimmt werden. So kann man z.B. festlegen, ob die Adresszeile, die Symbolleiste oder Scrollbalken angezeigt werden sollen. Eine Übersicht der möglichen Optionen finden Sie in Tabelle 14.1. Dort sind nur die Angaben aufgeführt, die sowohl vom Internet Explorer als auch von Netscape definiert werden können. Netscape kennt noch einige andere Angaben.

Optionale Angaben für ein neues Fenster

ANGABE	WERT	ERKLÄRUNG
`height`	`Pixel`	Höhe des Fensters
`left`	`Pixel`	Position links
`location`	`yes/no`	Adresszeile anzeigen
`menubar`	`yes/no`	Menüzeile anzeigen
`resizable`	`yes/no`	Größenänderung zulassen
`scrollbars`	`yes/no`	Scrollbalken anzeigen
`status`	`yes/no`	Statuszeile anzeigen
`toolbar`	`yes/no`	Symbolleiste anzeigen
`top`	`Pixel`	Position oben
`width`	`Pixel`	Breite des Fensters

Tabelle 14.1: Mit diesen Angaben kann man das Aussehen eines neues Fensters beeinflussen. Die Angaben werden von Internet Explorer, Netscape und Opera definiert. Lediglich die Angaben `top` und `left` kann Opera nicht auswerten.

✓ Tipp:

- Sie können die optionalen Angaben weglassen. Als Standardwert für die Optionen mit `yes/no` wird jeweils `no` verwendet. Wenn man für ein Fenster also keine Angabe zu `location` macht, wird das neue Fenster ohne Adresszeile angezeigt.

```
function e5(Datei,FName,Aussehen) {
NeuesFenster = win-
dow.open(Datei,FName,Aussehen);
}
```

Listing 14.14: Die Funktion selbst ist ganz einfach aufgebaut

```
<html>
<head>
<title>Neues Fenster</title>
<script src="dom.js"></script>
</head>
<body>
<a href="#" onClick="e5('../
seite2.html','w1', 'scrollbars=no,resiz-
able=yes,width=400,height=400,top=0,left=0
'); e5('http://www.mut.de','w2','scroll-
bars=no,resiz-
able=yes,width=400,height=400,top=0,left=4
00')">
Zwei neue Fenster</a>
</body></html>
```

Listing 14.15: Aufruf des Scripts in der Webseite. Die Angaben, die das Aussehen des Fensters beeinflussen, werden durch Kommas ohne Leerzeichen voneinander getrennt.

Die Funktion zum Öffnen eines Fensters

Die Funktion ist ganz einfach aufgebaut, da uns die Methode `open()` die Arbeit abnimmt (siehe Listing 14.14). Die Variable `NeuesFenster` ermöglicht es, vom Elternfenster aus auf das neue Fenster zuzugreifen (siehe hierzu auch die nachfolgenden Abschnitte).

In Listing 14.15 sehen Sie den Einsatz in einer Webseite. Mit einem Link wird die Funktion zweimal aufgerufen. Dadurch werden zwei neue Fenster erzeugt.

1. Als erster Parameter `Datei` wird die URL übergeben, die im neuen Fenster aufgerufen werden soll. Beim ersten neuen Fenster ist das die Datei *seite2.html* im übergeordneten Verzeichnis. Es kann auch eine absolute URL übergeben werden, z.B. *http://www.mut.de*, wie beim zweiten neuen Fenster.
2. Der zweite Parameter `FName` ist der Name des neuen Browserfensters. Wie bei allen selbst vergebenen Namen muss man auch bei neuen Fenstern darauf achten, keine reservierten Namen und Wörter zu verwenden. Die Bezeichnungen `_top`, `_self`, `_blank` und `_parent` können zwar verwendet werden. Sie sollten dies jedoch vermeiden, da diese Bezeichnungen in HTML eine vordefinierte Bedeutung haben.
3. Als dritter Parameter `Aussehen` werden die Optionen zum Aussehen des Browserfensters übergeben. Die Angaben werden dabei durch Kommas ohne Leerzeichen voneinander getrennt; die Reihenfolge ist dabei unerheblich.

Ein Fenster verschieben

Das Objekt `window` stellt zwei Methoden zum Verschieben eines Fensters zur Verfügung.

- Mit `moveBy()` wird ein Fenster um bestimmte Werte verschoben. Zwei Parameter müssen übergeben werden: Der erste entspricht dem Wert, um den das Fenster horizontal verschoben wird, der zweite Parameter ist der Wert, um den das Fenster nach unten oder oben verschoben wird. Positive Werte verschieben das Fenster nach rechts bzw. unten, negative Werte nach oben bzw. links.
- `moveTo()` verschiebt das Fenster zu einem bestimmten Punkt. Die neue Position gilt für den Ursprung des Fensters, also die linke obere Ecke. Auch hier müssen zwei Parameter übergeben werden. Der erste Wert legt die neue Startposition von links fest, der zweite Wert die neue Position von oben.

Da nicht alle Browser beim Öffnen von neuen Fenstern die Angaben `top` und `left` unterstützt, eignet sich das Verschieben mit `moveTo` z.B. zum Positionieren von neu geöffneten Fenstern.

In Abbildung 14.10 und Abbildung 14.11 sehen Sie ein Beispiel.

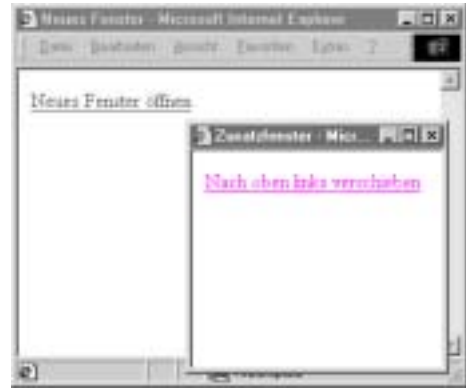

Abbildung 14.10: Das Popup-Fenster nach dem Öffnen

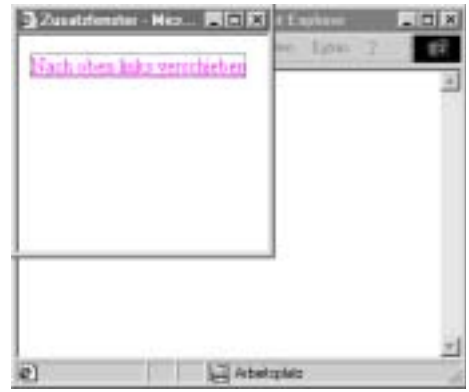

Abbildung 14.11: Das Fenster wurde nach links oben verschoben.

```
function e6(x,y) {
window.moveBy(x,y);
}
```

Listing 14.16: Mit `moveBy` wird ein Fenster um einen bestimmten Wert verschoben.

```
function e7(x,y) {
window.moveTo(x,y);
}
```

Listing 14.17: `moveTo` verschiebt ein Fenster an einen bestimmten Punkt.

```
function e6Kind(x,y) {
NeuesFenster.moveBy(x,y);
NeuesFenster.focus()
}
```

Listing 14.18: Die abgewandelte Funktion `moveBy`.

```
function e7Kind(x,y) {
NeuesFenster.moveTo(x,y);
NeuesFenster.focus()
}
```

Listing 14.19: Die abgewandelte Funktion `moveTo`.

```
<html><head>
<title>Fenster verschieben</title>
<script src="dom.js"></script>
</head><body>
<a href="#" onClick="e7(0,0);">
Nach oben links verschieben</a>
</body>
</html>
```

Listing 14.20: `moveTo` in einer Webseite. Beim Anklicken des Links wird das Fenster auf die Positionen `left` 0px und `top` 0px verschoben.

Funktionen zum Verschieben eines Fensters

Auch diese Funktionen sind einfach aufgebaut (siehe Listing 14.16 und Listing 14.17).

Wenn ein Zusatzfenster vom Elternfenster aus gesteuert werden soll, müssen die Funktionen etwas abgewandelt werden (siehe Listing 14.18 und Listing 14.19). Wenn mehrere Fenster geöffnet wurden, kann man nur auf das Fenster zugreifen, das zuletzt geöffnet wurde.

In Listing 14.20 sehen Sie den Einsatz von `moveTo` in einer Webseite, die in einem Zusatzfenster geöffnet wurde. Beim Anklicken des Links wird das Fenster auf die vorgegebene Position verschoben. Im Beispiel sind für die Positionen links und oben jeweils 0 Pixel angegeben, das Fenster wird also in die linke obere Ecke geschoben. Das Ergebnis konnten Sie bereits in Abbildung 14.10 und Abbildung 14.11 sehen.

Wenn das Fenster vom Elternfenster aus gesteuert werden soll, wird im Elternfenster die Funktion `e6Kind` bzw. `e7Kind` verwendet.

Ein Fenster schließen

Wenn man mit Popup-Fenstern arbeitet, sollte man dem Besucher auch immer einen Link zum Schließen des Fensters anbieten. Hierfür stellt das Objekt `window` die Methode `close()` zur Verfügung. Zum Schließen eines Fensters werden wieder zwei verschiedene Funktionen verwendet.

Die Funktion in Listing 14.22 schließt das Fenster, in dem sie aufgerufen wird. Wenn es sich dabei um ein Fenster handelt, das mit der `open()`-Methode geöffnet wurde, wird das Fenster ohne Nachfrage geschlossen. Wenn es sich um ein „normales“ Fenster handelt, erfolgt bei Internet Explorer und Netscape eine Sicherheitsabfrage (siehe Abbildung 14.12). Sie kann nicht unterdrückt werden.

Die Funktion in Listing 14.23 wird eingesetzt, wenn ein Zusatzfenster vom Elternfenster aus geschlossen werden soll. Auch hier gilt: Wenn mehrere Fenster geöffnet wurden, erfolgt der Zugriff nur auf das zuletzt geöffnete Fenster.

```
function e8() {
window.close()
}
```

Listing 14.21: Diese Funktion schließt das Fenster, in dem sie aufgerufen wird.

```
function e8Kind() {
NeuesFenster.close();
}
```

Listing 14.22: Diese Funktion schließt vom Elternfenster aus das Zusatzfenster.

```
<html><head>
<title>Zusatzfenster</title>
<script src="dom.js"></script>
</head><body>
<a href="#" onClick="e8();">
Das Fenster schließen</a>
</body></html>
```

Listing 14.23: Das Fenster wurde mit `open()` geöffnet, der Link schließt es wieder.

Abbildung 14.12: Die Sicherheitsabfrage in Netscape

15

WEITERE TECHNIKEN

In diesem Kapitel wird unter anderem der Umgang mit Frames behandelt. Um auf Objekte in Frames zuzugreifen, benötigt man eine angepasste Version des CrossBrowser-DOM.

Ferner lernen Sie fortgeschrittene Techniken wie die Animation von Objekten kennen. Einige Effekte verwenden das Crossbrowser-DOM, andere beruhen auf reinem JavaScript.

Ob Sie die Scripts nur in einzelnen Webseiten verwenden, gesammelt in einer neuen *.js*-Datei speichern oder die Datei *dom.js* erweitern, bleibt Ihnen überlassen und hängt auch davon ab, welche der Scripts Sie häufiger benötigen. Die meisten Beispiele in diesem Kapitel wurden in der Datei *effekte.js* gespeichert.

Allgemeines zu Frames

Frames teilen ein Browserfenster in Rahmen auf. In jedem dieser Rahmen wird eine andere Webseite aufgerufen, sodass mehrere Dokumente gleichzeitig geöffnet sind (siehe Abbildung 15.1).[1]

Abbildung 15.1: Die Webseite ist mit Frames in Rahmen aufgeteilt.

Der Browser behandelt jeden Frame als eigenes Fenster. Das Objekt `frames` ist eine Variante des Objekts `window`. Alle Eigenschaften und Methoden, die das Objekt `window` kennt, können auch auf das Objekt `frames` angewendet werden. Lediglich die Adressierung der Objekte in den Frames lautet anders.

Bisher verwendeten wir die Kurzform der DOM-Adresse. Statt `window.document.all` wurde z.B. `document.all` verwendet
(siehe hierzu auch Kapitel 11).

Der Zugriff auf Objekte in Frames erfolgt über den Name des Frames. Wenn ein Frame z.B. den Namen `menu` hat, erfolgt der Zugriff über `parent.menu.document.all` oder `top.menu.document.all`.

`parent` und `top` sind reservierte Fensternamen, die zum Ansprechen eines Frames im Frameset dienen.

Um auf die Objekte in den Frames zugreifen zu können, muss also eine angepasste Version des CrossBrowser-DOM erstellt werden.

Frames: Ja oder Nein?

Vor einigen Jahren war beim Einsatz von Frames noch Vorsicht geboten, weil die Technik damals neu war und viele Browser damit nicht umgehen konnten. Das ist heute nicht mehr der Fall.

Das weiteres Argument, das von Frame-Gegnern ins Feld geführt wird, ist der Platzbedarf. Natürlich benötigen Frames Platz, aber wenn sie sinnvoll eingesetzt werden, z.B. um die Navigation unterzubringen, ist auch dieses Argument hinfällig. Der Platz für die Navigation wird auch benötigt, wenn sie in jede Seite einzeln eingebunden wird. In einem Frame bleibt die Navigation auch sichtbar, wenn bei langen Seiten gescrollt werden muss und warum sollte man die Elemente der Navigation auf jeder Seite immer wieder erneut laden lassen?

Fazit: Dem Einsatz von Frames spricht nichts entgegen.

1. Es wird vorausgesetzt, dass Sie mit der Frame-Technik vertraut sind. Lesen Sie ggf. in einer HTML-Dokumentation nach.

```
function frameDOM(Stil,ObjID,Rahmen) {
if (idDOM) {
    if(Stil == 1) {
    return (parent[Rahmen].document.
    getElementById(ObjID).style);
    }
    else {
    return (parent[Rahmen].document.
    getElementById(ObjID));
    }
}

if (allDOM) {
    if(Stil == 1) {
    return (parent[Rahmen].
    document.all[ObjID].style);
    }
    else {
    return (parent[Rahmen].
    document.all[ObjID]);
    }
}

else {
    if (layDOM) {
    return (parent[Rahmen].
    document.layers[ObjID]);
    }
}

}
//Ende der Funktion
```

Listing 15.1: Die angepasste Funktion `frameDOM` in der Datei *framedom.js*.

Das DOM für Objekte in Frames

Zur Anpassung des CrossBrowser-DOMs wurde eine Kopie der Datei *dom.js* unter dem Namen *framedom.js* gespeichert. Anschließend wurde die Funktion `crossDOM` in `frameDOM` umbenannt und in der Funktion die erforderlichen Anpassungen vorgenommen.

1. `frameDOM(Stil,ObjID,Rahmen)`
   Die Parameter der Funktion müssen um den Wert für den Namen des Frames erweitert werden. Der neue Parameter wurde `Rahmen` genannt.
2. `parent[Rahmen].document.…`
   Die `return`-Anweisungen, die die DOM-Adressierung enthalten, werden jeweils um `parent[Rahmen].` erweitert. Beachten Sie, dass auch beim ID-DOM die eckigen Klammern gesetzt werden und nicht die runden Klammern wie bei der Objekt-ID.

Damit ist die Anpassung der Funktion `frameDOM` bereits fertig (siehe Listing 15.1).

Nun müssen noch die anderen Funktionen angepasst werden. Damit in einer Webseite bei Bedarf beide DOM-Dateien verwendet werden können, erhalten die veränderten Funktionen in der Datei *framedom.js* den Zusatz `Fr`, damit es nicht zu Überschneidungen bei den Funktionsnamen kommt, was zu Fehlermeldungen führen würde. Funktionen, die nicht angepasst wurden, sind in der Datei *framedom.js* gelöscht worden.

Funktionen mit crossDOM anpassen

In den Funktionen, in denen ursprünglich `crossDOM` angesprochen wurde, muss für die Variablen nun `frameDOM` gesetzt werden. Zudem muss als zusätzlicher Parameter der Wert `Rahmen` übergeben werden.

Als Beispiel sehen Sie in Listing 15.2 die Funktion `f8Fr` und `f9Fr`. Wenn die Funktionen in einer Frame-Seite aufgerufen werden, wird der Name des Frames, in dem das Objekt angesprochen werden soll, mit übergeben (siehe Listing 15.3).

In Abbildung 15.2 sehen Sie ein Frameset, bei dem Objekte im rechten Frame ein- und ausgeblendet werden, indem im linken Frame die entsprechenden Links angeklickt werden.

```
/* Für Objekte den Status visible setzen
*/
function f8Fr(ObjID,Rahmen) {
dom = frameDOM(1,ObjID,Rahmen)
dom.visibility = 'visible' ;
}

/* Für Objekte den Status hidden setzen */
function f9Fr(ObjID,Rahmen) {
dom = frameDOM(1,ObjID,Rahmen)
dom.visibility = 'hidden' ;
}
```

Listing 15.2: Die angepassten Funktionen

```
<a class="menu" href="#"
onClick="f8Fr('bild1','rechts');
f9('bild2','rechts');
f9('bild3','rechts');">Bild 1</a>
```

Listing 15.3: Der Aufruf der angepassten Funktionen in der Webseite

Abbildung 15.2: Ein- und Ausblenden von Objekten in einem anderen Frame

```
/* Breite der Anzeigefläche */
function f15Fr(Rahmen) {
if(parent[Rahmen].innerWidth != null) {
return (parent[Rahmen].innerWidth); }
if(parent[Rahmen].document.body.
clientWidth != null) {
return (parent[Rahmen].document.body.
clientWidth); }
else { return (null); }
}
```

Listing 15.4: Angepasste Funktion mit direkter DOM-Adressierung

```
<html><head>
<title>Frames</title>
<script src="framedom.js"></script>
<script language="JavaScript">
<!--
function anzeigen(Rahmen) {

breite = f15Fr(Rahmen);
alert('Der rechte Frame ist ' +breite+ 'px
breit.'); }

//--></script></head>
<body><p>
<a href="#" onClick="anzeigen('rechts')">
Breite des rechten Frames</a> </p>
</body></html>
```

Listing 15.5: Aufruf der angepassten Funktion

Andere Funktionen anpassen

Funktionen, in denen DOM-Adressen direkt verwendet werden, müssen durch Hinzufügen von `parent[Rahmen].` angepasst werden.

Als Beispiel sehen Sie in Listing 15.4 die angepasste Funktion `f15Fr`. Aus der DOM-Adresse `window.innerWidth` wurde `parent[Rahmen].innerWidth`, aus `document.body.clientWidth` wird `parent[Rahmen].document.body.clientWidth`.

Auch hier wird beim Aufruf der Funktion der Name des entsprechenden Frames mit übergeben (siehe Listing 15.5).

✓ Tipps:

- Die Funktionen `e5` bis `e8Kind` müssen nicht geändert werden, da sie sich auf neue Fenster und nicht auf die Frames beziehen. Sie wurden in der Datei *framedom.js* gelöscht und stehen bei Bedarf durch *dom.js* zur Verfügung.
- Auch einige andere Funktionen wurden nicht angepasst, da man sie standardmäßig nur im gleichen Dokument verwendet, z.B. `f13`, mit der Bilder abhängig von der Farbtiefe angezeigt werden.

Besonderheiten bei den Funktionen

Die Rollover-Funktion e4 stellt eine Ausnahme dar. Normalerweise werden Rollovers nur im gleichen Fenster verwendet. In diesem Fall kann weiterhin die Funktion e4 benutzt werden.

Wenn man aber einen Frame-übergreifenden Rollover-Effekt realisieren will, muss eine neue Funktion programmiert werden, da dann die Variable Rahmen erstellt werden muss. Sie wurde e4Fr genannt (siehe Listing 15.6).

Wie der Aufruf der neuen Funktion in der Webseite erfolgt, sehen Sie in Listing 15.7.

Abbildung 15.3 und Abbildung 15.4 zeigen ein Beispiel für einen Frame-übergreifenden Rollover-Effekt.

```
/* Frame-übergreifender Rollover-Effekt*/
function e4Fr() {
if (document.images) {
Rahmen = e4Fr.arguments[2];
parent[Rahmen].document[e4Fr.argu-
ments[0]].src=e4Fr.arguments[1];
}
}
```

Listing 15.6: Die neue Rollover-Funktion

```
<a href="#" onmouseo-
ver="e4Fr('button01','images/
button01_over.gif','rechts');" onmouse-
out="e4Fr('button01','images/
button01.gif','rechts');">Home</a>
```

Listing 15.7: Als dritter Parameter wird der Name des Rahmens übergeben.

Abbildung 15.3: Die Seite nach dem Laden.

Abbildung 15.4: Im linken Frame wurde der Link Listings mit der Maus überfahren, im rechten Frame ändert sich dabei die Grafik.

```
if (top.location.href != location.href) {
top.location.href = location.href;
}
```

Listing 15.8: Das Script befreit eine Seite aus fremden Framesets.

✓ Tipp:

- Wenn Sie selbst ein Frameset verwenden, binden Sie das Script nur in die Datei ein, die das Frameset enthält. In den Inhaltsseiten, die in Ihrem Frameset aufgerufen werden, sollten Sie das Script zum Erzwingen des Framesets einbinden, das auf der nächsten Seite vorgestellt wird.

Tipps zu Frames

Seiten aus einem fremden Frameset befreien

Wenn aus einem fremden Frameset heraus auf Ihre Webseite verwiesen wird und das fremde Frameset beim Anklicken des Links nicht aufgelöst wird, erscheint Ihre Seite im fremden Frameset. Dies kann ohne böse Absicht geschehen, wenn beim Link das Attribut `target=_blank` (oder `target=_top`) vergessen wurde. Es gibt aber auch unfaire Zeitgenossen, die versuchen, fremde Inhalte als Teil der eigenen Website darzustellen.

Es ist nicht wünschenswert, wenn die eigenen Inhalte in fremden Framesets auftauchen. Wenn Sie selbst ein Frameset verwenden, wird außerdem das Aussehen der Seite stark beeinträchtigt. Frame-übergreifende Scripts funktionieren ggf. auch nicht mehr.

Eine Seite lässt sich ganz einfach aus einem fremden Frameset befreien. Verwenden Sie dazu das kleine Script aus Listing 15.8.

Anzeige im eigenen Frameset erzwingen

Wenn Sie ein Frameset verwenden und direkt auf eine Ihrer Inhaltsseiten verlinkt wird, wird nur diese Seite angezeigt, nicht aber das Frameset, das fast immer die Navigation enthält. Mit dem Script aus Listing 15.9 können Sie sicherstellen, dass die Webseite immer in Ihrem Frameset angezeigt wird.

Anstelle von `FrameName` setzen Sie den Namen des entsprechenden Frames ein und statt `frameset.html` den Dateinamen Ihres Framesets. Achten Sie darauf, dass für diese Datei der richtige relative Pfad gesetzt wird, wenn die Inhaltsseite in einem anderen Verzeichnis gespeichert ist. Sie können auch die absolute URL Ihrer Website verwenden.

Wichtig: Das Script funktioniert in manchen Browsern nicht, wenn bei den Links, mit denen die Seiten aufgerufen werden, das Attribut `target` verwendet wird. Wenn die Seiten immer im gleichen Frame aufgerufen werden, können Sie im `<head>`-Bereich der Seite das Tag `<base target="AnzeigeFrame">` verwenden. Wenn die Dateien in unterschiedlichen Frames aufgerufen werden, verwenden Sie die Funktion, die im nachfolgenden Abschnitt vorgestellt wird.

```
if (window.name != 'FrameName'){
top.location.href="frameset.html";
}
```

Listing 15.9: Das Script erzwingt die Anzeige einer Seite im eigenen Frameset.

✓ Tipps:

- Das Script würde auch mit der Bedingung `if(top.location.href == location.href)` funktionieren, dann wäre es aber möglich, die Inhaltsseiten direkt in einem fremden Frameset aufzurufen. Der Browser würde in solchen Fall feststellen, dass sich die Seite in einem Frameset befindet und das Script nicht ausführen. Dass es sich dabei um ein fremdes Frameset handelt, kann der Browser nicht wissen.
- Geben Sie Ihren Frames auf jeden Fall Namen, die nicht geläufig sind. Wenn Sie als Framename `menu`, `haupt`, `main` o.ä. verwenden, wird die Seite zwar in einem Frameset angezeigt, das muss dann aber nicht Ihr eigenes sein. Falls aus einem fremden Frameset heraus auf eine Inhaltsseite verlinkt wird, und ein Frame des fremden Framesets zufällig den gleichen Namen hat, würde die Seite stattdessen im fremden Frameset aufgerufen werden. Geeignet sind Framenamen, die sich direkt auf Ihre Website beziehen, z.B. `cssunddhtml_menu` für den Navigationsframe.

Abbildung 15.5: Die Inhalte der beiden rechten Frames werden gleichzeitig geändert.

```
function z1(Seite,Rahmen) {
parent[Rahmen].document.location.href =
Seite;
}
```

Listing 15.10: Beachten Sie, dass der Zeilenumbruch durch die Spaltenbreite im Buch bedingt ist. Im Quellcode ist er nicht vorhanden.

```
<a href="java-
script:z1('beispiel1.html','beispiel');
z1('quelle1.html','quelle')">
Beispiel 1</a>
```

Listing 15.11: Der mehrfache Aufruf der Funktion im Browser

Mehrere Frames gleichzeitig ändern

In HTML besteht keine Möglichkeit, mit einem Klick die Inhalte von mehreren Frames gleichzeitig zu ändern. Mit JavaScript ist das kein Problem.

In Abbildung 15.5 sehen Sie eine Webseite, in der links die Navigation steht, rechts oben wird ein Beispiel angezeigt, darunter der jeweilige Quellcode für das Script. Wenn in der Navigation ein Link angeklickt wird, werden die Inhalte der beiden rechten Rahmen gleichzeitig geändert.

Die Funktion ist sehr einfach aufgebaut (siehe Listing 15.10).

In der Webseite wird für jeden Frame, der geändert werden soll, die Funktion mit Angabe der URL und des entsprechenden Framenamens aufgerufen (siehe Listing 15.11).

✓ Tipp:

- Dieses Script eignet sich auch, wenn nur der Inhalt eines Frames geändert werden soll. Wenn Sie die Anzeige der Seiten im Frameset erzwingen (siehe Listing 15.9), können Sie das Script verwenden, um die Browserbugs zu umgehen, die beim Erzwingen des Framesets auftreten können, wenn das Attribut `target` verwendet wird.

Ein Objekt animieren

Wie man ein Objekt in eine andere Position verschiebt, wurde bereits im letzten Kapitel besprochen. Das ist aber nicht das, was man unter einer Animation versteht. Bei einer Animation besteht ein zeitlicher Ablauf, der am Bildschirm beobachtet werden kann.

Ein Beispiel sehen Sie in Abbildung 15.6 bis Abbildung 15.8. In diesem Beispiel wandert ein weißer Spot über die Webseite und macht den Schriftzug "Willkommen" sichtbar.

Der Spot besteht aus einer GIF-Grafik mit transparentem Hintergrund, in deren Mitte ein weißer Kreis gesetzt wurde. Die Grafik wurde als positioniertes Objekt definiert. Beim Laden der Seite wird das Objekt animiert und die Grafik wandert über den Bildschirm.

Der Schriftzug hat die gleiche Farbe wie der Hintergrund der Webseite und wurde ebenfalls als Objekt positioniert. Der z-Index des Schriftzugs ist höher als der der Grafik. Dadurch wandert der Spot in der Ebene unterhalb der Schrift über den Bildschirm und macht den Schriftzug sichtbar.

Abbildung 15.6: Von links...

Abbildung 15.7: ...über die Mitte...

Abbildung 15.8: ...nach rechts

```
function z2Set(ObjID,Ex,Ey,Sx,Sy) {
EndeX = Ex;
EndeY = Ey;
SprungX = Sx;
SprungY = Sy;
//zeit = 50;
dom = crossDOM(1,ObjID);
aktuellX = f4(ObjID);
aktuellY = f3(ObjID);
z2(aktuellX,aktuellY);
}
```

Listing 15.12: In dieser Funktion werden die Variablen für die Animation erstellt und die eigentliche Animation gestartet.

✓ Tipps:

- Die Zeit wird in Millisekunden angegeben, ein Wert von `1000` bedeutet also, dass das Zeitintervall eine Sekunde beträgt. Je höher der Wert, desto langsamer die Animation. Für die meisten Animationen ist ein Wert von 20 bis 100 angemessen.
- Die Geschwindigkeit der Animation wird nicht nur durch das Zeitintervall beeinflusst. Auch der Sprungwert spielt eine Rolle. Wenn das Zeitintervall auf 20ms eingestellt wird und der Sprungwert auf 3px, läuft die Animation schneller ab als bei einem Sprungwert von 1px.
- Gleichzeitig beeinflusst der Sprungwert natürlich auch den flüssigen Ablauf der Animation. Bei höheren Werten ruckelt die Animation.

Programmierung der Animation

Die Animation besteht aus zwei Funktionen. In der ersten Funktion `z2Set` werden die erforderlichen Variablen erstellt und dann wird die eigentliche Animation gestartet, die in der Funktion `z2` programmiert wird.

Die Setup-Funktion

1. `z2Set(ObjID,Ex,Ey,Sx,Sy)`
   Folgende Parameter werden übergeben: die Objekt-ID `ObjID`; die Positionswerte `Ex` (horizontal) und `Ey` (vertikal), bei denen die Animation gestoppt wird und die Sprungwerte `Sx` und `Sy`. Mit diesen Sprungwerten legen Sie fest, um wie viele Pixel das Objekt pro Zeitintervall bewegt wird.
2. `Variablen`
   Die Variablen werden erstellt. Wenn für alle Animationen immer das gleiche Zeitintervall verwendet werden soll, können die Kommentarzeichen vor der Variablen `zeit` gelöscht werden. Im Beispiel wurde die Variable `zeit` in der Webseite gesetzt; in der Funktion ist sie daher auskommentiert. Die aktuellen Positionen für die Variablen `aktuellX` und `aktuellY` werden aus den Funktionen `f4` und `f3` bezogen.
3. `z2(aktuellX,aktuellY);`
   Die Funktion `z2` wird mit den Positionswerten gestartet, die in der CSS-Definition für das Objekt gesetzt wurden.

Die Setup-Funktion finden Sie in Listing 15.12.

Die Animationsfunktion

In der Funktion z2 wird die eigentliche Animation programmiert. In dieser Funktion werden ständig die aktuellen Positionswerte ermittelt und mit den angegeben Stopp-Positionen verglichen. Auch die Zeitsteuerung der Animation wird in dieser Funktion überwacht.

1. `if(aktuellX != EndeX)`
   Mit den `if`-Bedingungen im oberen Teil der Funktion wird geprüft, ob die Stopp-Positionen erreicht wurden. Wenn die aktuelle Position nicht mit der Stopp-Position übereinstimmt, wird geprüft, ob der aktuelle Wert größer ist als die Stopp-Position. Ist das der Fall, wird der aktuelle Wert um den Sprungwert verkleinert. Damit werden Objekte von rechts nach links bzw. von unten nach oben über die Anzeigefläche bewegt. Die `else`-Anweisungen in den Verzweigungen gelten, wenn der Wert der aktuellen Position kleiner ist als die Stopp-Position. Der aktuelle Wert wird dann um den Sprungwert erhöht und das Objekt wird von links nach rechts bzw. von oben nach unten bewegt.
2. `if((aktuellX!=EndeX) || (aktuellY!=EndeY))`
   Wenn eine der aktuellen Positionen nicht mit der Stopp-Position übereinstimmt, wird für das Objekt die Position `left` bzw. `top` neu gesetzt.

Fortsetzung auf der nächsten Seite

```
function z2(aktuellX,aktuellY) {
//Stopp-Position erreicht?
if(aktuellX != EndeX) {
if(aktuellX > EndeX) {
aktuellX = aktuellX - SprungX; }
else {
aktuellX = aktuellX + SprungX; }
}
if(aktuellY != EndeY) {
if(aktuellY > EndeY) {
aktuellY = aktuellY - SprungY; }
else {
aktuellY = aktuellY + SprungY; }
}

/*Wenn die Stopp-Position noch nicht
erreicht ist, wird zeitgesteuert jeweils
eine neue aktuelle Position gesetzt:*/
if((aktuellX != EndeX) ||
(aktuellY != EndeY)) {
if (domStil.pixelLeft) {
domStil.pixelLeft = aktuellX;
domStil.pixelTop = aktuellY;
}
else {
domStil.top = aktuellY;
domStil.left = aktuellX;
}
setTimeout ('z2(' +aktuellX+ ','
+aktuellY+ ')',zeit);
}
}
```

Listing 15.13: Die Funktion für die eigentliche Animation

```
<body onLoad="z2Set('kreis',700,25,5,0)">
```

Listing 15.14: Aufruf der Setup-Funktion.

```
<script language="JavaScript">
<!--
zeit = 20;
//-->
</script>
```

Listing 15.15: Die Variable `zeit` wird im Head-Bereich der Seite gesetzt.

```
#kreis { position:absolute; top:25px;
left:-140px; margin:0px; padding:0px; z-
index:0; }
#schrift { position:absolute; top:60px;
font:bold 36px Arial,sans-serif;
color:#000000; margin:0px; padding:0px; z-
index:1; }
```

Listing 15.16: Definition der beiden Objekte

✓ Tipps:

- Um ein Objekt von rechts nach links zu bewegen, wird in der CSS-Definition eine `top`-Position gesetzt, die außerhalb des rechten Bildschirmrands liegt, z.B. 1200px. In der Funktion wird als Stopp-Position ein negativer Wert angegeben.
- Animationen, die außerhalb der Anzeigefläche beginnen oder enden, sollten Sie in einem Fenster aufrufen, dessen Größe sie kontrollieren. Dazu können Sie entweder ein neues Browserfenster öffnen, oder die Anzeigefläche beim Laden der Seite auf die gewünschten Werte bringen (siehe nächster Abschnitt).

3. `setTimeout('z2('+aktuellX+',
   '+aktuellY+')',zeit);`
   Die Methode `setTimeout` kommt ebenfalls nur zum Einsatz, wenn die Stop-Position noch nicht erreicht ist. `setTimeout` führt die Funktion `z2` erneut aus. Die Variable `zeit` bestimmt dabei, in welchem Zeitintervall der Funktionsaufruf erfolgt. Bei der Methode `setTimeout` müssen die Parameter der Funktion `z2` als Zeichenketten übergeben werden.

Listing 15.13 zeigt die Funktion für die Animation.

Einsatz der Animation in der Webseite

Im Beispiel (siehe die Abbildung 15.6 bis Abbildung 15.8) wurde die Animation durch den Event-Handler `onLoad` im `<body>`-Tag gestartet. Die Funktion `z2Set` wird aufgerufen. Dabei werden die erforderlichen Parameter übergeben (siehe Listing 15.14).

Da die Variable `zeit` im Script auskommentiert wurde, um mehrere Animationen mit unterschiedlichen Zeitintervallen starten zu können, wird die Variable im `<head>`-Bereich der Seite gesetzt (siehe Listing 15.15).

Die CSS-Definition der Objekte `kreis` und `schrift` finden Sie in Listing 15.16.

Das Objekt `kreis` wird animiert, bis die Position `left` bei `700px` liegt. Der horizontale Sprungwert beträgt dabei `5px`. Die End-Position für den `top`-Wert entspricht dem Ausgangswert, da das Objekt nur horizontal bewegt werden soll. Der vertikale Sprungwert wurde daher auf `0px` gesetzt.

Die Größe der Anzeigefläche ändern

Wenn ein Besucher Ihre Webseite aufruft, wissen Sie nie, wie groß das Browserfenster in diesem Moment ist. Auf die Größe der Anzeigefläche können Sie aber Einfluss nehmen, und auf diese Fläche kommt es eigentlich an. Das Objekt `window` stellt die Methoden `resizeBy()` und `resizeTo()` zur Verfügung.

`resizeBy()` ändert die Größe der Anzeigefläche um einen bestimmten Wert. Alle gängigen Browser beziehen `resizeBy` auf die Anzeigefläche. Manche Browser berücksichtigen dabei die Scrollbalken, andere nicht. Dennoch ist die Größe der Anzeigefläche bei der Verwendung der Methode `resizeBy` annähernd gleich.

Anders sieht es bei der Methode `resizeTo()` aus. Mit `resizeTo` bringt man das Fenster auf eine feste Zielgröße. Das ist genau das, was wir erreichen wollen. Leider definieren die Browser den Begriff Fenster dabei völlig unterschiedlich. Netscape 4 ändert die Anzeigefläche auf die Zielgröße, die anderen Browser ändern das komplette Browserfenster inklusive aller Leisten. Die Größe der Anzeigefläche nach der Änderung ist daher sehr unterschiedlich, je nach Browser und abhängig davon, ob Andockleisten, Symbolleisten usw. vorhanden sind.

Wie unterschiedlich die Ergebnisse je nach Browser mit `resizeTo` ausfallen, sehen Sie in Abbildung 15.9 und Abbildung 15.10. Eine einheitlich große Anzeigefläche erhalten wir so jedenfalls nicht.

Statt dessen programmieren wir eine Funktion, in der wir `resizeBy` verwenden.

Abbildung 15.9: Für die Höhe wurden bei `resizeTo` 300 Pixel angegeben. Hier ist das unterschiedliche Ergebnis deutlich zu sehen.

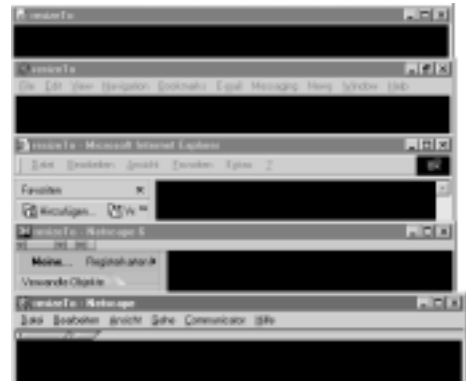

Abbildung 15.10: Auch die Breite der Anzeigefläche ist unterschiedlich, wenn Andockfenster vorhanden sind.

```
function z3(ZB,ZH) {
zielB = ZB;
zielH = ZH;
aktuellB = f15();
aktuellH = f14();
maxBreite = screen.width;
if(maxBreite > 790) {
aendernB = zielB - aktuellB;
aendernH = zielH - aktuellH;
window.resizeBy(aendernB,aendernH);
}
}
```

Listing 15.17: Die Funktion ändert die Größe der Anzeigefläche auf bestimmte Werte.

Größe der Anzeigefläche

Bedenken Sie bei der gewünschten Zielhöhe, dass viele Anwender immer noch eine Auflösung von 800x600 Pixel verwenden und ggf. etliche Symbolleisten eingeblendet haben. Die Titel- und Statusleiste sollten auch berücksichtigt werden. Internet Explorer und Opera bringen es dabei auf ca. 200 Pixel, wenn alle Leisten eingeblendet sind. Sie sollten daher die Zielhöhe der Anzeigefläche für diese Auflösung auf max. 400 Pixel setzen.

Die Anwender sind daran gewöhnt, dass eine Seite senkrecht gescrollt werden muss. Waagrechtes Scrollen hingegen wird kaum akzeptiert und veranlasst sogar viele Besucher dazu, die Website nicht mehr aufzurufen. Die Breite der Anzeigefläche ist daher noch wichtiger als die Höhe. Mehr als 750 Pixel sollten Sie für die Breite der Anzeigefläche nicht ins Auge fassen.

Die Funktion zur Änderung der Größe der Anzeigefläche

1. `function z3(ZB,ZH)`
   Als Parameter für die neue Funktion `z3` werden die Werte für die gewünschte Größe der Anzeigefläche übergeben. In der Funktion werden die Variablen `zielB` (Breite) und `zielH` (Höhe) für die Parameter erstellt.
2. `aktuellB=f15();`
   `aktuellH=f14();`
   Wenn die Anzeigefläche mit `resizeBy()` auf eine bestimmte Größe gebracht werden soll, muss bekannt sein, wie groß die Anzeigefläche nach dem Öffnen der Seite tatsächlich ist. Diese Werte kann man mit den Funktionen `f14` und `f15` auslesen.
3. `maxBreite=screen.width;`
   `if(maxBreite>790)`
   Die Bildschirmauflösung wird berücksichtigt. Nur wenn die Breite der Auflösung größer ist als 790px, wird die Größe der Anzeigefläche geändert. Die Werte, um die die Anzeigefläche geändert werden soll, werden errechnet, indem die aktuelle Größe von der Zielgröße abgezogen wird. Wenn die aktuelle Anzeigefläche größer ist als die gewünschte Zielgröße, werden negative Werte errechnet und das Fenster wird verkleinert. Positive Werte vergrößern das Fenster.

Sie finden die Funktion in Listing 15.17.

Automatischer Zuschlag für hohe Bildschirmauflösungen

In der Funktion z4 erhält die Höhe der Anzeigefläche einen automatischen Zuschlag von 100px, wenn die Bildschirmauflösung höher als 800x600 ist (siehe Listing 15.18).

Die Breite der Anzeigefläche wird nicht geändert. Ein Seitenlayout sollte in der Breite auf ca. 740px begrenzt sein, damit bei der Auflösung 800x600 keine waagerechten Scrollbalken auftreten. Eine Änderung der Breite macht also wenig Sinn.

Aufruf der Funktionen in der Webseite

Der Aufruf für beide Scripts ist mit Ausnahme des Funktionsnamens identisch. Sinnvoll ist der Einsatz des Event-Handlers onLoad im <body>-Tag. Damit wird die Anzeigefläche beim Aufruf der Seite auf die gewünschte Größe gebracht.

Als erster Parameter wird die gewünschte Breite angegeben, als zweiter Wert die Höhe (siehe Listing 15.19).

```
function z4(ZB,ZH) {
zielB = ZB;
zielH = ZH;
aktuellB = f15();
aktuellH = f14();
maxBreite = screen.width;
if(maxBreite > 790) {
if(maxBreite > 810) {
aendernH = zielH - aktuellH +100;
}
else {
aendernH = zielH - aktuellH;
}
aendernB = zielB - aktuellB;
window.resizeBy(aendernB,aendernH);
}
}
```

Listing 15.18: Die Höhe der Anzeigefläche erhält einen automatischen Zuschlag von 100px, wenn höhere Bildschirmauflösungen als 800x600 verwendet werden.

```
<html><head>
<title>resizeBy</title>
<script src="dom.js"></script>
<script src="effekte.js"></script>
</head>
<body onLoad="z4(750,400);">
</body></html>
```

Listing 15.19: Aufruf der Funktion mit dem Event-Handler onLoad

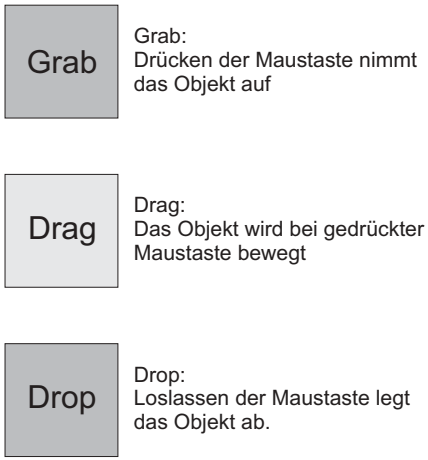

Abbildung 15.11: Drei Vorgänge sind erforderlich. Eigentlich müsste es also Grab, Drag and Drop heißen.

Drag&Drop von Objekten

Was früher nur der Mac konnte, ist inzwischen bei allen grafisch orientierten Betriebssystemen eine Selbstverständlichkeit: Drag&Drop von Objekten.

Drag&Drop besteht nicht aus zwei, sondern aus drei Vorgängen (siehe Abbildung 15.11). Für jeden dieser Vorgänge benötigen wir eine Funktion:

1. *Grab*: Das angeklickte Objekt wird identifiziert und erhält den z-index 1000.
2. *Drag*: Das Objekt wird abhängig von der Mausposition bewegt.
3. *Drop*: Die DOM-Variable wird auf Null zurückgesetzt, damit bei Bedarf ein anderer Drag&Drop-Vorgang ausgeführt werden kann. Der z-index des Objekts wird auf 500 gesetzt.

Jede dieser Funktionen ist an ein Mausereignis gekoppelt:

- Grab: `onMouseDown`
- Drag: `onMouseMove`
- Drop: `onMouseUp`

Die Funktionen sind abhängig von der Mausposition. Also benötigen wir auch noch eine Funktion, die die Mausereignisse überwacht und mit den Drag&Drop-Funktionen verbindet. Die vier Funktionen wurden in der Datei *dragdrop.js* gespeichert.

Anforderungen an die Webseite

Damit das Script in allen gängigen Browsern funktioniert, muss die Webseite für den verschiebbaren Bereich einige Voraussetzungen erfüllen:

- Eine verschiebbare Ebene darf kein Kindelement einer anderen Ebene sein. Sie muss mit dem `<div>`-Tag direkt im `<body>` der Seite eingebunden werden.
- In der Ebene muss mindestens eine Grafik vorhanden sein.
- Der Name der Ebene muss einmal als ID im `<div>`-Tag angegeben werden und zusätzlich als Name im `<img>`-Tag der Grafik.

In Abbildung 15.12 und Abbildung 15.13 sehen Sie ein Beispiel für ein verschiebbares Menü. Die Menügrafik ist aus mehreren Einzelgrafiken aufgebaut; das Symbol zum Verschieben des Bereichs wurde mit dem Namen der Ebene versehen.

✓ Tipp:

- Verschiebbare Bereiche eignen sich gut für Menüs und DHTML-Spiele.

Abbildung 15.12: Das Verschieben-Symbol signalisiert dem Besucher, dass der Bereich verschoben werden kann.

Abbildung 15.13: Das Menü kann auf der Webseite frei verschoben werden.

```
function ereignisse() {

if(layDOM) {
document.captureEvents (Event.MOUSEDOWN |
Event.MOUSEMOVE | Event.MOUSEUP )
}
if(DHTML) {

document.onmousedown = grab;
document.onmousemove = drag;
document.onmouseup = drop;
}
}
```

Listing 15.20: Die Mausereignisse werden den Funktionen zugeordnet.

```
<body onLoad="ereignisse();">
```

Listing 15.21: Aufruf der Mausüberwachung im Body der Seite durch `onLoad`

Überwachung der Mausereignisse

Zur Überwachung von Ereignissen steht das Objekt `event` zur Verfügung. `event` ist ein Unterobjekt von `window`. Leider gehen die Browser damit völlig unterschiedlich um. Netscape kennt für `event` andere Eigenschaften als der Internet Explorer, auch die Syntax ist verschieden.

Die Zuordnung der Ereignisse ist noch einfach. Mit `document.onmousedown = grab;` wird z.B. dem Event `onMouseDown` die Funktion `grab` zugewiesen. Beachten Sie, dass hier nur der Name der Funktion verwendet wird, die runden Klammern mit den Parametern werden nicht angegeben.

Netscape 4 benötigt zusätzlich die Methode `captureEvents`, um die Ereignisse zu überwachen. Achten Sie hier bitte auf die Schreibweise. Mehrere Events werden durch ein einzelnes Pipe-Zeichen | voneinander getrennt.

Die Funktion zur Mausüberwachung finden Sie in Listing 15.20. Sie wird im `<body>`-Tag mit dem Event-Handler `onLoad` aufgerufen (siehe Listing 15.21).

Die Funktion grab

Der Begriff *grab* kommt aus dem Englischen und bedeutet im Computerjargon, dass ein Objekt identifiziert und an den Mauscursor angeheftet wird.

Diese Funktion ist bei weitem die komplexeste des Drag&Drop-Effekts. Hier müssen etliche Verzweigungen programmiert werden, damit die gängigen Browser das Objekt identifizieren können. Die Browser benutzen teilweise eine völlig unterschiedliche Syntax.

Opera 5/6 kommt mit dem Script nur zurecht, wenn der Browser als Internet Explorer angemeldet ist. Daher geben wir Opera-Nutzern, die eine andere Browser-Identifikation eingestellt haben, einen entsprechenden Hinweis durch eine `alert`-Meldung (siehe Abbildung 15.14).

Sie können den Text natürlich jederzeit ändern. Weisen Sie aber bitte in jedem Fall darauf hin, dass Opera als Internet Explorer angemeldet sein muss. In Listing 15.23 (siehe ab nächster Seite) steht statt der tatsächlich verwendeten Meldung nur ein Platzhalter, um das Listing kürzer zu halten. Globale Variablen

Zuerst müssen zwei globale Variablen erstellt und auf `null` gesetzt werden: `domStil` und `name` (siehe Listing 15.22).

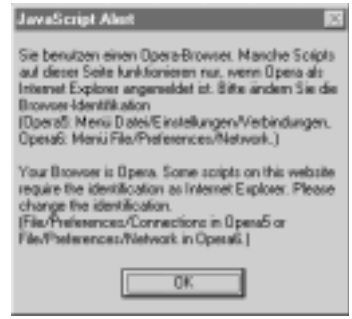

Abbildung 15.14: `Alert`-Meldung für Opera-Nutzer, wenn die Browser-Identifikation nicht auf Internet Explorer eingestellt ist

```
/* Globale Variablen domStil und name
erstellen + auf Null setzen: */
var domStil = null;
var name = null;
```

Listing 15.22: Die globalen Variablen `domStil` und `name` müssen erstellt und auf `null` gesetzt werden.

```
/*Grab: Das angeklickte Objekt
wird identifiziert*/

function grab(Ereignis) {

//Opera als IE angemeldet?

if (window.opera && (IE != 1)) {
alert("…Meldung einsetzen…");
}

/* Wenn Opera als IE angemeldet oder der
Browser kein Opera ist: */

else {
//Abfrage für IE

if (IE) {
ObjID = event.srcElement.name;
if (ObjID){
domStil = crossDOM(1,ObjID);
}
}
```

Listing 15.23: Fortsetzung auf der nächsten Seite

Das Listing zur Funktion grab

1. `function grab(Ereignis)`
   Der Parameter `Ereignis` muss für Netscape übergeben werden. Netscape verwendet nicht unmittelbar das Objekt `event` (siehe weiter unten).
2. `if (window.opera && (IE != 1))`
   Zuerst wird geprüft, ob der verwendete Browser Opera ist. Wenn die Browser-Identifikation nicht auf IE eingestellt ist, wird der Hinweis als `alert`-Meldung ausgegeben (siehe Abbildung 15.14).
3. `else`
   In der nachfolgenden `else`-Anweisung wird nun auf die verschiedenen Browser geprüft.
4. `if(IE)`
   Internet Explorer, Opera und IE-kompatible Browser identifizieren das Objekt direkt über `event.srcElement.name`, wenn das Ereignis eintritt.
5. `if(layDOM)`
   `var findeObjekt;`
   `var x = Ereignis.pageX;`
   `var y = Ereignis.pageY;`
   Bei Netscape 4 ist der Aufwand bedeutend größer. Der Browser erkennt zwar, an welcher Stelle die Maustaste gedrückt wurde, wenn der Parameter `Ereignis` übergeben wird, kann das Objekt aber nicht unmittelbar identifizieren. Die Mausposition verwenden wir für die lokalen Variablen `x` und `y`. Außerdem brauchen wir noch die lokale Variable `findeObjekt`.

Fortsetzung auf der nächsten Seite

6. for-Schleife
   N4 speichert alle Ebenen im Array layers. Dieses Array wird mit der for-Schleife durchlaufen, dabei wird mit der Bedingung document.layers.length-1 rückwärts gegen 0 gezählt. Warum Anzahl der Layer minus 1? Im Array layers werden die Ebenen nach ihrem Auftreten im HTML-Quelltext abgelegt und per Index verwaltet. Die erste im Dokument vorkommende Ebene wird also als layers[0] gespeichert, die nächste als layers[1] usw. bis hin zur letzten Ebene.Wenn zehn Ebenen vorhanden sind, wird die letzte Ebene als layers[9] verwaltet. Setzt man für die Anzahl der Ebenen e, ist die letzte Ebene also als layers[e-1] gespeichert. Die letzte Ebene liegt im natürlichen z-index auch gleichzeitig oben. Stellen Sie sich diese zehn Ebenen als Folienstapel vor. Wenn Sie mit einem Stift auf dem Stapel schreiben, können Sie immer nur die Folie beschriften, die oben liegt. Um an die darunter liegenden Folien zu gelangen, müssen Sie den Stapel von oben her Folie für Folie abtragen. Genau das macht auch die for-Schleife, indem rückwärts gezählt wird.
7. In der for-Schleife werden durch die ellenlange if-Bedingung für jeden Layer die Abmessungen berechnet und mit den Koordinaten der Mausposition aus den Variablen x und y verglichen. Liegt der Mausklick innerhalb einer Ebene, wird die (Layer-)DOM-Adresse dieser Ebene an die Variable domStil übergeben.

Fortsetzung auf der nächsten Seite

Fortsetzung

```
//Abfrage für N4

if(layDOM) {

var findeObjekt;
var x = Ereignis.pageX;
var y = Ereignis.pageY;

for (var i = document.layers.length -1; i
>=0; i--) {
findeObjekt = document.layers[i];

if((x > findeObjekt.left) && (x < finde-
Objekt.left + findeObjekt.clip.width) &&
(y > findeObjekt.top) && (y < findeOb-
jekt.top + findeObjekt.clip.height)) {
domStil = findeObjekt;
}
}
}
```

Listing 15.23: Fortsetzung auf der nächsten Seite

Fortsetzung

```
//Abfrage für N6

if (window.netscape ) {
ObjID = Ereignis.target.name;
if (ObjID){
domStil = crossDOM(1,ObjID);
}
}
//anderer Browser mit idDOM?
else {
ObjID = event.srcElement.id;
if (ObjID){
domStil = crossDOM(1,ObjID);
}
}
}

//z-index auf 1000 setzen
if(domStil) {
domStil.zIndex = 1000;
return false;
}
/*Wenn kein Objekt gefunden wurde, domStil
wieder auf null setzen*/
else {
domStil = null;
return;
}
}
//Ende der grab-Funktion
```

Listing 15.23: Die Funktion `grab`

8. Nachdem Netscape 4 uns soviel Arbeit gemacht hat, geht es bei Netscape 6 wieder recht schnell. Im Gegensatz zu N4 kann N6 über `window.netscape` identifiziert werden. Wie IE/Opera erkennt N6 sofort das angeklickte Objekt; die Syntax ist allerdings unterschiedlich. N6 kann die Ebene nicht direkt über das Objekt `event` erkennen, sondern benötigt den Parameter `Ereignis`. Die Referenzierung des Objekts erfolgt über `Ereignis.target.name`.
9. Für andere Browser mit ID-DOM steht die `else`-Anweisung mit `srcElement.id`.
10. Wenn das Objekt identifiziert wurde, wird der z-Index auf `1000` gesetzt, um die Ebene in jedem Fall in den Vordergrund zu holen, auch wenn sehr viele Objekte vorhanden sind.
11. Wenn kein Objekt gefunden wurde, wird die Variable `domStil` wieder auf Null gesetzt.

Geschafft! Keine Sorge, die anderen Scripts sind bedeutend kürzer.

Die Funktion drag

Im Gegensatz zur Funktion `grab` ist die Funktion `drag` angenehm übersichtlich.

1. `function drag(Ereignis)`
   Auch hier muss für Netscape der Parameter `Ereignis` übergeben werden.
2. `if(domStil)`
   Zuerst wird geprüft, ob in der Variablen `domStil` ein Wert gespeichert ist. Falls dies der Fall ist, wird das Script ausgeführt.
3. `if(NN)`
   Netscape 4 und 6 arbeiten zum Ermitteln der Cursorposition mit `Ereignis.pageX` bzw. `Ereignis.pageY`. Abhängig von der Position des Mauscursors werden die Startpositionen `top` und `left` geändert.
4. `else`
   Bei den anderen Browsern wird die Position `left` mit `window.event.x` geändert; `top` mit `window.event.y`.

Sie finden die Funktion in Listing 15.24.

```
function drag(Ereignis) {

if(domStil) {

if(NN) {
domStil.left = Ereignis.pageX;
domStil.top = Ereignis.pageY;
}

else {
domStil.left = window.event.x;
domStil.top = window.event.y;
}
}
}
//Ende Funktion drag
```

Listing 15.24: Die Funktion drag ändert abhängig von der Position des Mauscursors die Startposition `top` und `left`.

Die Funktion drop

Mit dieser Funktion wird lediglich der `z-Index` geändert und die Variable `domStil` wieder auf `null` gesetzt. Die Übergabe des Parameters `Ereignis` ist daher nicht erforderlich. Das kurze Script bedarf wohl keiner Erklärung. Sie finden es in Listing 15.25.

```
function drop() {
if(domStil) {
domStil.zIndex = 500;
domStil = null;
}
}
```

Listing 15.25: Die Funktion `drop`

```
<html><head><title>Drag&Drop</title>
<script src="dom.js"></script>
<script src="dragdrop.js"></script>
<style type="text/css"><!--
#grafikb { position:absolute; top:30px;
left:30px; border:0px;}
#grafikc { position:absolute; top:30px;
left:65px; border:0px;}
#grafikd { position:absolute; top:30px;
left:100px; border:0px;}
--></style></head>
<body onLoad="ereignisse();">
<div id="grafikb"><img name="grafikb"
src="b.gif" border="0"></div>
<div id="grafikc"><img name="grafikc"
src="c.gif" border="0" alt=""></div>
<div id="grafikd"><img name="grafikd"
src="d.gif" border="0" alt=""></div>
</body></html>
```

Listing 15.26: Einfaches Drag&Drop-Beispiel

Abbildung 15.15: Links sehen Sie die Webseite nach dem Laden. Rechts wurden die Grafiken per Drag&Drop verschoben.

Die Webseite

Dass die Funktion `ereignisse()` im `<body>`-Tag mit `onLoad` aufgerufen wird, wurde bereits im Abschnitt zu dieser Funktion erwähnt.

Da `ereignisse()` die Mausereignisse überwacht, müssen die Funktionen `grab`, `drag` und `drop` im Quelltext der Webseite nicht eingebunden werden.

In Listing 15.26 sehen Sie den Quelltext für eine einfache Webseite. Beachten Sie, dass die Ebenen nicht in anderen Ebenen als Kindelement erstellt werden dürfen. Der Name der Ebenen wird mit dem Attribut `id` im `<div>`-Tag und mit dem Attribut `name` im `<img>`-Tag notiert.

Die Grafiken können einzeln per Drag&Drop verschoben werden (siehe Abbildung 15.15).

Komplexere Grafiklayouts

Normalerweise gebe ich dem Positionieren durch CSS den Vorzug, für das Menü in Abbildung 15.16 wurde aber eine blinde Tabelle verwendet.

Der Grund hierfür ist die Tatsache, dass die Menügrafik in mehrere Grafiken gesplittet wurde, da für die Buttons Rollover-Effekte verwendet wurden. Das Positionieren der Teilgrafiken ist mit CSS kein Problem, das Verschieben des ganzen Menüs wäre dann aber nicht mehr möglich. Es würden immer nur die Teilgrafiken verschoben werden.

Die Menügrafik könnte auch als Imagemap verwendet werden. Viele Browser haben jedoch Probleme, wenn innerhalb einer Imagemap mehrere verschiedene Funktionen verwendet werden.

Bei komplexen Grafiklayouts greift man daher am besten auf die althergebrachte Methode der blinden Tabellen zurück.

Abbildung 15.16: Die Menügrafik wurde in mehrere Teilgrafiken gesplittet. Diese Grafiken sind in einer blinden Tabelle angeordnet.

16

NAVIGATION IN DER WEBSITE

Die Navigationsmöglichkeiten innerhalb einer Website sind sehr wichtig. Eine gute Navigation bestimmt darüber, ob die Besucher auf Ihrer Website bleiben und sie erneut aufsuchen.

Navigationsleisten nehmen Platz ein. In diesem Kapitel werden einige Lösungen vorgestellt, die Ihnen helfen, wertvollen Platz für die Inhalte Ihrer Website zu sparen. Außerdem erhalten Sie Tipps, was Sie bei der Navigation grundsätzlich beachten sollten.

Was macht eine gute Navigation aus?

Es gibt einige Punkte, die man bei der Navigation beachten sollte. Manches wird Ihnen selbstverständlich erscheinen, aber wenn man mitten in einem Projekt steckt, kann man diese Aspekte leicht aus dem Auge verlieren.

Planung

Eine professionelle Navigation beginnt schon bei der Planung einer Website. Dabei geht es vorrangig nicht um das Aussehen der Navigation oder der Seiten, sondern um die Inhalte der Präsenz und die Aufteilung in bestimmte Rubriken.

Nehmen Sie sich ein Blatt Papier und schreiben Sie auf, welche Inhalte Ihre Website haben soll. Ordnen Sie die Inhalte nach Bereichen und geben Sie diesen Rubriken Namen. Das kann in Form von Listen geschehen, ebenso können Sie dafür auch Flowcharts verwenden (siehe Abbildung 16.1).

Diese Aufteilung bestimmt die Struktur der Navigation und dient gleichzeitig als Grundlage für die Sitemap der Website. Die Ordnerstruktur der Website sollten Sie ebenfalls nach dieser Aufteilung erstellen.

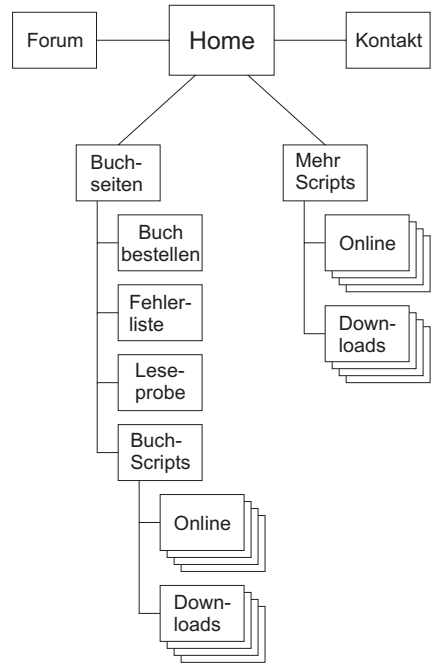

Abbildung 16.1: Die Seitenstruktur bildet die Grundlage der Navigation.

Abbildung 16.2: Hier ist es nicht erforderlich, dass die Untermenüs Links zu den anderen Rubriken enthalten, da das Hauptmenü stets oben waagerecht zu sehen ist.

Abbildung 16.3: Dieses Untermenü enthält auch die Links zu den anderen Rubriken.

Struktur der Navigation

- Das Hauptmenü enthält die Links zu den Rubriken. Es macht keinen Sinn, auf jeder Seite die Links zu allen anderen Seiten einzubinden. Bei umfangreichen Websites ist das auch gar nicht möglich. Die Rubriken sollen aber von jeder Seite aus erreichbar sein.
- Jede Rubrik erhält ein Untermenü, das die Inhalte dieser Rubrik anzeigt. Wenn Unterrubriken vorhanden sind, erhalten auch diese ein eigenes Menü. Das mag auf den ersten Blick aufwändig erscheinen, wenn man den Quelltext für die Menüs aber in eigenen Dateien speichert, kann man sie einfach per Copy&Paste in die Seiten einbinden.
- Ob man in die Untermenüs auch die Links zu den anderen Rubriken einbindet, hängt davon ab, wie die Navigation aufgebaut ist (siehe die Abbildung 16.2 und Abbildung 16.3).

✓ Tipps:

- Viele Editoren bieten die Möglichkeit, die Menüs als so genannte Bausteine zu speichern und bei Änderungen am Baustein alle Seiten, in denen er vorhanden ist, automatisch zu aktualisieren. Sie müssen also nur den Baustein ändern und nicht jede einzelne Seite. Zu diesen Editoren gehören u.a. Dreamweaver, aber auch mein persönlicher Favorit Phase5, ein HTML-Editor für Windows, der unter der URL *http://www.meybohm.de* zum kostenlosen Download bereitsteht.
- Wenn Ihre Website auf einem Server gehostet ist, auf dem Sie Server Side Includes (SSI) verwenden können, verwenden Sie die Menübausteine als Includes. Informationen zu SSI finden Sie auf der Webseite zum Buch beim Script für das Dropdown-Menü.

Anforderungen an die Navigation

Die Startseite der Website kann ein anderes Aussehen haben als die Unterseiten, so wie ein Buchcover ein völlig anderes Layout als die Kapitel hat. Eine gute Navigation sollte die folgenden Anforderungen erfüllen:

- *Beständigkeit:* Die Art und Position der Navigation sollte auf allen Seiten beibehalten werden. Wenn in einer Rubrik die Navigation auf der linken Seite zu finden ist, erwartet der Besucher das auch in den anderen Rubriken. Verwirren Sie Ihre Besucher nicht damit, dass Sie von einem Navigationsframe plötzlich auf Dropdown-Menüs wechseln.
- *Der Weg zur aktuellen Seite:* Zeigen Sie dem Besucher, wo er sich befindet. Bei einer Navigation, wie sie in Abbildung 16.3 gezeigt wurde, weiß der Besucher, in welcher Rubrik er sich befindet. Bei anderen Navigationsarten ist das nicht immer ersichtlich. Dann sollten Sie dem Besucher zeigen, wo er sich gerade befindet. Das erleichtert außerdem einen Wechsel in die übergeordnete Rubrik. Ein Beispiel sehen Sie in Abbildung 16.4.
- *Quicklink zur Navigation:* Wenn die Navigation nicht in einem Frame untergebracht ist, wird sie bei langen Seiten aus dem Blickfeld gescrollt. Bieten Sie einen Link an, der den Besucher zum Anfang der Seite und damit auch zur Navigation zurückbringt. Auch wenn die Navigation ausgeblendet werden kann (siehe Abbildung 16.4), muss die Möglichkeit vorhanden sein, das Menü jederzeit wieder aufzurufen.

Fortsetzung auf der nächsten Seite

Abbildung 16.4: Im oberen Bereich der Seite wird dem Besucher angezeigt, wo er sich gerade befindet.

Kein JavaScript?

Obwohl die meisten Surfer moderne Browser verwenden, besteht die Möglichkeit, dass der Anwender die Unterstützung für JavaScript abgeschaltet hat oder doch einen Browser verwendet, der mit JavaScript nicht umgehen kann.

Sie sollten daher auf allen Seiten, die JavaScript und CSS enthalten, einen `<noscript>`-Bereich einbinden. `<noscript>` funktioniert, weil Browser, die Scriptsprachen *unterstützen*, diesen Bereich komplett ignorieren. Wenn ein Browser Scriptsprachen *nicht unterstützt*, werden nur die Tags `<noscript>` und `</noscript>` ignoriert, derText dazwischen wird jedoch angezeigt.

Fassen Sie die wichtigsten Informationen Ihrer Website zusammen und verweisen Sie im `<noscript>`-Bereich auf diese alternativen Seiten. Wenn Ihre regulären Seiten auch ohne JavaScript funktionstüchtig sind, können Sie natürlich auch diese Seiten referenzieren.

Weisen Sie darauf hin, dass der Besucher mehr von Ihren Seiten hat, wenn er einen Browser mit JavaScript-Unterstützung verwendet.

Es besteht auch die Möglichkeit, bereits beim Aufruf der Startseite auf den Browser zu reagieren. Browser mit Unterstützung für JavaScript werden durch ein Script auf die regulären Seiten weitergeleitet, ältere Browser bleiben auf der Startseite, die dann nur die Links zu den Seiten ohne JavaScript enthält.

Fortsetzung

- *Vor-/Zurück-Links:* Viele Besucher nutzen die Vor- und Zurück-Schalter des Browsers, um bereits gelesene Seiten noch mal aufzurufen. Es gibt aber auch Anwender, die die Symbolleiste ausgeblendet haben, damit für die Webseite mehr Anzeigefläche zur Verfügung steht. Nicht zu vergessen sind die Internet-Einsteiger, die oft noch nicht wissen, wozu diese Browserbuttons dienen. Bieten Sie eigene Links an, die den gleichen Zweck erfüllen wie die Browserbuttons.
- *Alternative Navigation:* Bieten Sie für Browser, die JavaScript nicht unterstützen, eine alternative Navigation an. Binden Sie dazu einen `<noscript>`-Bereich mit Links zu Seiten ein, die die wichtigsten Informationen Ihrer Website enthalten. Siehe hierzu auch den Infokasten rechts.
- *Navigation im Frame:* Hier gilt das Gleiche wie bei JavaScript: Bieten Sie alternative Links an. Da die Browser mit dem `<noframe>`-Bereich jedoch sehr unterschiedlich umgehen, sollten Sie die alternativen Links nicht im `<noframe>`-Tag anbieten. Setzen Sie nach dem Frameset einfach den `<body>`-Container und notieren Sie die Links dort. Der `<body>`-Bereich wird nur angezeigt, wenn ein Browser mit Frames nicht umgehen kann. Sorgen Sie außerdem dafür, dass die Seiten auch wirklich im Frameset aufgerufen werden (siehe hierzu *Tipps zu Frames* in Kapitel 15).

Vor-/Zurück-Links

Links, die wie die Browserbuttons in den aufgerufenen Seiten vor- und zurückblättern, werden mit Hilfe des Objekts `history` realisiert. `history` ist ein Unterobjekt von `window` und erlaubt einen sehr beschränkten Zugriff auf die Historyliste des Besucher-Browsers. Aus Sicherheitsgründen kann man die History weder löschen noch die URLs der besuchten Seiten auslesen.

Die Eigenschaft `length` speichert die Anzahl der besuchten Seiten. Als Methoden stehen `back()`, `forward()` und `go(n)` zur Verfügung. Der Parameter `n` bei der Methode `go` ist ein Platzhalter für die Anzahl der Seiten, um die gesprungen werden soll. Positive Werte springen dabei vor, negative Werte springen zurück.

In Abbildung 16.5 sehen Sie Quellcode-Beispiele für solche Links.

Abbildung 16.5: Links mit dem Objekt `history`

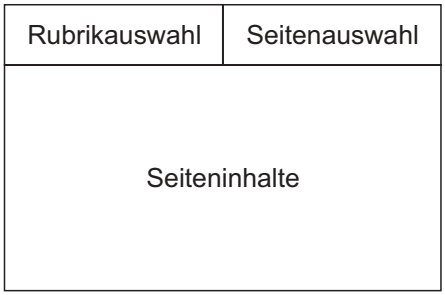

Abbildung 16.6: Aufteilung des Framesets

Abbildung 16.7: Links wurde die Rubrik Buch gewählt, rechts erscheint die Auswahlliste für die Rubrik.

Abbildung 16.8: Diesmal wurde der Bereich für die zusätzlichen Scripts gewählt. Im rechten Frame wird die entsprechende Liste aufgerufen.

Navigation mit Auswahllisten

Eine der Platz sparendsten Möglichkeiten, in der man zusätzlich sehr viele Links unterbringen kann, ist die Navigation mit Auswahllisten. Sie ist außerdem eine gute Alternative zu Ihrer normalen Navigation, die der Besucher bei Bedarf zuschalten kann.

Grundgerüst für diese Navigation ist ein Frameset mit drei Frames (siehe Abbildung 16.6).

Abhängig von der Rubrik, die der Besucher in der linken Liste auswählt, wird im rechten oberen Frame die Auswahlliste für die gewählte Rubrik aufgerufen (siehe die Abbildung 16.7 und Abbildung 16.8). Nach der Auswahl der gewünschten Seite in der rechten Liste wird das entsprechende Dokument im unteren Frame aufgerufen.

Das Frameset

1. Zuerst wird das Fenster in zwei Zeilen aufgeteilt. Die obere Zeile, die die Auswahllisten enthält, ist nur `40`px hoch. Die restliche Fläche wird durch das Sternchen `*` dem Rahmen für die Seiteninhalte zugewiesen.
2. Die Zeile für die Auswahllisten wird in zwei Spalten aufgeteilt. Der linke Frame ist `200`px breit und erhält den Namen `rubrikwahl`. Beim Start des Framesets wird dort die Datei `rubrik.html` aufgerufen. Sie enthält die Liste mit den verschiedenen Rubriken.
3. Die restliche Breite der oberen Zeile wird dem Frame `seitenwahl` zugewiesen. Beim Start des Framesets wird dort die Datei `blind.html` aufgerufen. Sie enthält nur das HTML-Grundgerüst, aber keine Inhalte. Je nach ausgewählter Rubrik wird hier später das entsprechende Rubrikmenü aufgerufen.
4. Der Frame für die Seiteninhalte heißt `inhalte`, dort wird anfangs die eigentliche Startseite der Website aufgerufen, im Beispiel hat sie den Namen `start.html`.

In Listing 16.1 sehen Sie den Quelltext zum Frameset.

```
<html><head><title>Navigation mit Auswahl-
listen</title></head>
<frameset rows="40,*" border="0"
frameborder="0" framespacing="0">
<frameset  cols="200,*">
<frame scrolling="no" name="rubrikwahl"
src="rubrik.html" marginheight="3"
marginwidth="10">
<frame scrolling="no" name="seitenwahl"
src="blind.html" marginheight="3"
marginwidth="10">
</frameset>
<frame scrolling="auto" name="inhalte"
src="start.html">
</frameset>
<body>Ihr Browser unterstützt leider keine
Frames. </body></html>
```

Listing 16.1: Der Quelltext des Framesets

```
<body>
<form action="" name="form1">
<select  size=1 name="Auswahl"
onChange="waehle(this.form.Auswahl.options
[this.form.Auswahl.options.selected-
Index].value)" class="Auswahl"
width="190">
<option value="nichts">
[ Rubrik wählen: ]</option>
<option value="nichts">
--------------------</option>
<option value="index.html">
Home</option>
<option value="buch/navi.html">
Buch</option>
<option value="forum.html">
Forum</option>
<option value="kontakt.html">
Kontakt</option>
<option value="mehr/navi.html">
Noch mehr Scripts?</option>
<option value="nichts">
--------------------</option>
<option value="ende">
Beenden</option></select>
</form></body></html>
```

Listing 16.2: `<body>`-Bereich der Datei rubrik.html

Die Rubrikauswahl

In Listing 16.2 sehen Sie den Quellcode für den `<body>`-Bereich der Datei *rubrik.html*. Diese Datei wird im linken oberen Frame aufgerufen, sie stellt die Rubriken zur Auswahl. Den Head-Bereich mit dem Script sehen wir uns später an. Zuerst befassen wir uns mit den Werten, die später im Script verwendet werden.

1. `<form action="" name="form1">`
   Die Auswahlliste muss in ein `<form>`-Tag eingebettet werden. Das Attribut `action` bleibt leer, als Formularname wurde `form1` festgelegt.
2. `<select  size=1 name="Auswahl"`
   Mit `select` wird die Auswahlliste eingeleitet, `size=1` legt fest, dass nur eine Zeile des Felds angezeigt wird. Der Name des Formularfelds ist `Auswahl`.
3. `onChange="waehle(this.form.Auswahl.options[n].value)"`
   Mit dem Event-Handler `onChange` wird bei jeder Auswahl aus der Liste die Funktion `waehle` aufgerufen, die im Head der Seite notiert wurde (mehr dazu später). Als Parameter wird der Wert `value` der gewählten Option `option[n]` übergeben. Für `n` wird normalerweise die Indexnummer der gewählten Option übergeben. Da die Funktion aber bei jeder Auswahl aufgerufen wird, übergeben wir stattdessen `this.form.Auswahl.options.selectedIndex`.
4. `value="nichts"`
   Bei jeder Option wird das Attribut `value` angegeben. Auf die entsprechenden Werte des Attributs wird im Script reagiert.

Das Auswahlscript

Im <head>-Bereich der Seite wird die Funktion waehle() definiert (siehe Listing 16.3).

1. function waehle(x)
   Als Parameter wird x angegeben. Da beim Aufrufen der Funktion im Formular als Parameter der value-Wert der gewählten Option übergeben wird, z.B. nichts oder index.html, entspricht x also diesem übergegebenen Wert.
2. if(x=="nichts")
   Der Wert nichts steht für Optionen, bei denen keine Änderung stattfinden soll. In diesem Fall wird lediglich das Formular zurückgesetzt.
3. if(x=="ende")
   Sie können dem Besucher die Möglichkeit geben, die Navigation zu beenden. Mit dem value-Wert ende wird im obersten Fenster top die Datei aufgerufen, die gerade im Frame inhalte als Seite angezeigt wird. Das Frameset wird damit aufgelöst; es ist nur noch die gewählte Inhaltsseite zu sehen.
4. if(x=="buch/navi.html")
   Für Rubriken, die mehr als eine Seite enthalten, wurden Unterverzeichnisse angelegt. Für solche Rubriken werden auch Rubrikmenüs benötigt. Im Frame inhalte wird die Startseite der jeweiligen Rubrik aufgerufen; im Frame seitenwahl der übergebene value-Wert. In diesem Fall ist dies die Auswahlliste für die Rubrik, aus der der Besucher die einzelnen Seiten auswählen kann.

Fortsetzung auf der nächsten Seite

```
<html><head><title>Rubriken</title>
<style type="text/css"><!--
.Auswahl { width:190px; font:9pt
Verdana,Arial,Helvetica,sans-serif; }
--></style>
<script type="text/javascript">
<!--

function waehle(x) {
if(x == "nichts") {
document.form1.reset();
return;
}
else if(x == "ende") {
top.location.href = parent.inhalte.
location;
}
else if(x == "buch/navi.html") {
parent.inhalte.location.href = "buch/
index.html";
parent.seitenwahl.location.href = x;
}
else if(x == "mehr/navi.html") {
parent.inhalte.location.href =
"mehr/index.html";
parent.seitenwahl.location.href = x;
}
else {
parent.inhalte.location.href = x;
parent.seitenwahl.location.href =
"blind.html";
}
}
//--></script></head>
```

Listing 16.3: Das Script für die Auswahl der Rubriken

```
<body>
<form action="" name="form1">
<select size=1 name="Auswahl"
onChange="waehle(this.form.Auswahl.options
[this.form.Auswahl.options.selected-
Index].value)" class="Auswahl"
width="200">
<option value="nichts">
[ Rubrik Buch ]</option>
<option value="nichts">
--------------------</option>
<option value="index.html">
Startseite der Rubrik Buch</option>
<option value="bestellen.html">
Das Buch bestellen</option>
<option value="referenz.html">
CSS-Referenz</option>
<option value="listings.html">
Die Buch-Scripts</option>
<option value="fehler.html">
Fehlerliste zum Buch</option>
<option value="nichts">
--------------------</option>
<option value="verzeichnis.html">
Inhaltsverzeichnis zum Buch</option>
<option value="probe.html">
Leseprobe</option>
<option value="nichts">
 </option>
</select>
</form></body></html>
```

Listing 16.4: Der Body eines Untermenüs

Fortsetzung

5. `else`
Bei Rubriken, für die nur eine Seite vorhanden ist, wird kein Rubrikmenü gebraucht. Im Frame `inhalte` wird die jeweilige Seite aufgerufen, die mit dem `value`-Wert übergeben wurde; im Frame `seitenwahl` die Datei `blind.html`. Wenn im rechten Frame eine Auswahlliste für eine andere Rubrik vorhanden war, wird sie dadurch ausgeblendet.

Die Menüauswahl der Rubriken

Die Dateien für die Rubrikmenüs sind im Prinzip ebenso aufgebaut wie die eben besprochene Datei *rubrik.html*. Als Beispiel wird die Datei *navi.html* aus dem Verzeichnis `buch` verwendet. Der Aufbau des `<body>`-Bereichs entspricht dabei der Datei *rubrik.html*, wobei bei den `value`-Werten der Optionen natürlich andere URLs übergeben werden und die Texte der Einträge anders sind. Außerdem wurde die Beenden-Option weggelassen und als letzter Eintrag wurde eine Leerzeile eingefügt.

Sie finden den `<body>`-Bereich der Datei `buch/navi.html` in Listing 16.4.

Die Auswahlfunktion des Rubrikmenüs

Auch die Funktion zum Wählen der Seiten ist in den Rubrikmenüs fast identisch mit dem Script in der Datei *rubrik.html*. DieVerzweigung zum Beenden wurde weggelassen, da sie hier nicht benötigt wird.

Im Beispiel sind auch keine weiteren Untermenüs erforderlich; daher wird nach der Bedingung `if(x=="nichts")` nur noch die `else`-Anweisung gesetzt, die die gewählte Seite im Frame `inhalte` aufruft. In Listing 16.5 finden Sie den `<head>`-Bereich der Datei `buch/navi.html`.

✓ Tipp:

- Wenn Sie innerhalb einer Rubrik ein Untermenü benötigen, teilen Sie die obere Zeile des Framesets in drei Spalten auf. Im Script des Rubrikmenüs rufen Sie dann mit einer Verzweigung, wie sie auch in der Datei *rubrik.html* verwendet wurde, das Untermenü für die Auswahl auf. Im Beispiel wäre das z.B. für die Auswahl `Buch-Scripts` denkbar. Im dritten oberen Frame könnten die Kapitelnummern angezeigt werden.

```
<html><head><title>
Navigation Rubrik Buch</title>
<style type="text/css">
<!--
.Auswahl { width:200px; font:9pt Ver-
dana,Arial,Helvetica,sans-serif; }
--></style>
<script type="text/javascript">
<!--

function waehle(x) {
if(x == "nichts") {
document.form1.reset();
return;
}
else {
parent.inhalte.location.href = x;
}
}
//--></script></head>
```

Listing 16.5: Der Head des Rubrikmenüs

Abbildung 16.9: Abhängig von der gewählten Rubrik wird unter der Linie die jeweilige Menügrafik eingeblendet.

Abbildung 16.10: In der unteren Grafik reagieren die Einträge ebenfalls als Rollover.

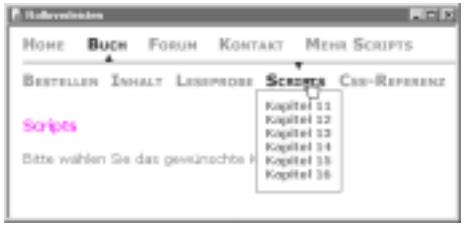

Abbildung 16.11: Wenn Untermenüs benötigt werden, kann das durch Einblenden von Ebenen realisiert werden.

Grafische Rollover-Leisten

Eine Lösung, die ebenfalls nicht viel Platz verbraucht, sind grafische Rolloverleisten, wie sie in Abbildung 16.9 und Abbildung 16.10 zu sehen sind. Diese Leisten können ohne Frameset realisiert werden, die Lösung mit Frames spart aber das erneute Laden der Grafiken.

1. Die Leiste besteht aus etlichen Teilgrafiken; mit einem Bildbearbeitungsprogramm wie Adobe ImageReady (Bestandteil von PhotoShop) ist diese Arbeit aber rasch erledigt. Sie können die Schriftzüge für alle Rubrikmenüs in einer Datei erstellen und die nicht benötigten Bestandteile ausblenden.
2. Die Grafiken für den unteren Teil der Leiste (die Rubrikmenüs) werden in Ebenen abgelegt. Je nach gewählter Rubrik werden diese Ebenen ein- und ausgeblendet.
3. Falls Untermenüs benötigt werden, können diese ebenfalls durch Einblenden entsprechender Ebenen realisiert werden. Wenn die Rollover-Leiste in einem Frame liegt, muss die Ebene für das Untermenü in die Inhaltsseite im unteren Frame eingebunden werden.

Die erforderlichen Techniken (Rollover sowie Ein- und Ausblenden von Ebenen) sind aus den vorangegangenen Kapiteln bekannt.

Auf der Webseite zum Buch finden Sie ein entsprechendes Beispiel mit den erforderlichen HTML-Dateien und einer PhotoShop-Datei, die die Grundlage für die benötigten Grafiken bildet.

Dynamische Frames

Wenn die Navigation in einem Frameset untergebracht ist und gerade nicht benötigt wird, kann man die Navigation über ein Script ausblenden und so den Seiteninhalten mehr Platz zur Verfügung stellen.

Damit das Script funktioniert, muss vor das Frameset, das für das Layout der Seite zuständig ist, ein weiteres Frameset gesetzt werden, das die Steuerung übernimmt. Mehr dazu auf den nächsten Seiten.

In Abbildung 16.12 sehen Sie die Aufteilung für das Frameset, das die Steuerung übernimmt. Abbildung 16.13 zeigt das normale Frameset für das Layout der Seite mit Navigationsmenü.

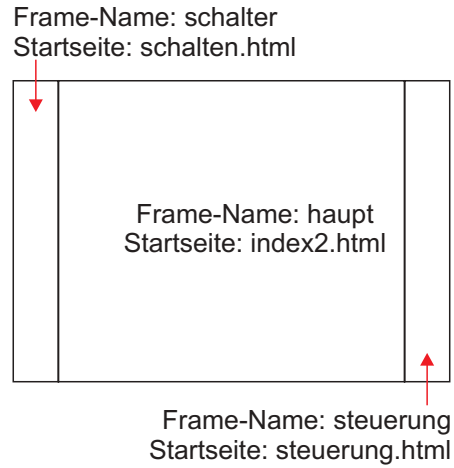

Abbildung 16.12: Die Datei index.html übernimmt die Steuerung zum Ein- und Ausblenden des Menüs.

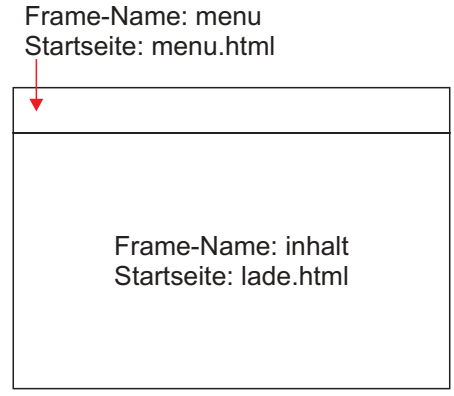

Abbildung 16.13: Das normale Frameset für das Seitenlayout hat den Namen mit_menu.html.

```
<html><head>
<title>Dynamische Frames</title>
<frameset  frameborder="0" border="0"
framespacing="0"  cols="30,*,1">
<frame  marginheight="0" marginwidth="0"
scrolling="no"  name="schalter"
src="schalten.html">
<frame  marginheight="0" marginwidth="0"
scrolling="auto" name="haupt"
src="mit_menu.html">
</frameset>
<frame  marginheight="0" marginwidth="0"
scrolling="no" name="steuerung"
src="steuerung.html">
<body>
Ihr Browser unterstützt leider keine
Frames!</body></html>
```

Listing 16.6: Das Frameset index.html

Die Datei index.html

Die Datei *index.html* ist die Startseite der Website. Sie enthält das Frameset, das die Steuerung zum Ein- und Ausblenden des Menüs übernimmt.

1. `cols="30,*,1"`
   Das Frameset ist in drei Spalten aufgeteilt. Der linke Rahmen ist 30px breit, der rechte Frame nur 1px. Die restliche Fläche zwischen den beiden Frames wird durch das Sternchen dem mittleren Frame zugeteilt.
2. Im Frame `schalter` wird die Datei *schalten.html* referenziert, die den Link zum Ein- und Ausblenden des Menüs enthält.
3. Im Frame `haupt` wird anfänglich das normale Frameset für das Seitenlayout mit Navigationsmenü aufgerufen. Diese Datei trägt den Namen *mit_menu.html*. Wenn der Navigationsframe ausgeblendet werden soll, wird im Frame `haupt` die Datei *ohne_menu.html* aufgerufen.
4. Der Frame `steuerung` enthält die Datei *steuerung.html*, in der das Script zum Aus- und Einblenden des Menüs enthalten ist. Internet Explorer und Netscape kommen mit dem Script auch zurecht, wenn es direkt in der Datei *schalten.html* notiert ist; Opera aber nicht, dort werden die Werte der Variablen teilweise nicht korrekt verarbeitet. Daher benötigt man die zusätzliche Datei. Sie hat keine weiteren Inhalte.

Den Quelltext der Datei `index.html` sehen Sie in Listing 16.6.

Bevor wir uns der Steuerung zuwenden, kommt zuerst noch das Frameset für das Seitenlayout und die darin aufgerufenen Dateien an die Reihe.

Die Datei mit_menu.html

Die Datei *mit_menu.html* ist ebenfalls ein Frameset und wird zunächst in der Datei *index.html* im Frame `haupt` aufgerufen. *mit_menu.html* ist für das Layout der Webseiten mit Navigationsframe zuständig.

1. Im Frame `menu` wird die Datei *menu.html* aufgerufen. Sie enthält das Navigationsmenü.
2. Im Frame `inhalt` wird die Datei *lade.html* aufgerufen. Diese Datei ist keine normale Inhaltsseite, sondern lädt über ein Script beim Einblenden des Menüs erneut die Seite, die bei ausgeblendetem Menü zuletzt aufgerufen war.

In Listing 16.7 steht der Quelltext der Datei *mit_menu.html*.

Die Datei ohne_menu.html

Auch die Datei *ohne_menu.html* ist ein Frameset. Sie wird in der Datei *index.html* im Frame `haupt` aufgerufen, wenn der Navigationsframe ausgeblendet werden soll.

1. `rows="1,*"`
   Der Aufbau ist identisch mit der Datei *mit_menu.html*, der Frame `menu` wurde hier aber auf `1px` verkleinert.
2. `name="menu" src="blind.html"`
   Im Frame `menu` wird diesmal eine leere Datei (*blind.html*) aufgerufen. Sie enthält nur das HTML-Grundgerüst, aber keinerlei anderen Inhalte.

In Listing 16.8 steht der Quelltext der Datei *ohne_menu.html*.

```
<html><head>
<title>Dynamische Frames</title>
<frameset  frameborder="0" border="0"
framespacing="0"  rows="50,*">
<frame  marginheight="0" marginwidth="0"
scrolling="no"  name="menu"
src="menu.html">
<frame  marginheight="0" marginwidth="0"
scrolling="auto" name="inhalt"
src="lade.html">
</frameset>
<body>
Ihr Browser unterstützt leider keine
Frames!</body></html>
```

Listing 16.7: Das Frameset mit_menu.html

```
<html><head>
<title>Dynamische Frames</title>
<frameset  frameborder="0" border="0"
framespacing="0"  rows="1,*">
<frame  marginheight="0" marginwidth="0"
scrolling="no"  name="menu"
src="blind.html">
<frame  marginheight="0" marginwidth="0"
scrolling="auto" name="inhalt"
src="lade.html">
</frameset>
<body>
Ihr Browser unterstützt leider keine
Frames!</body></html>
```

Listing 16.8: Das Frameset ohne_menu.html

```
<html><head><title>Menü</title>
<link rel="stylesheet" href="seiten.css"
type="text/css">
<style type="text/css"><!--
body { margin:5px; }
p { margin:0px; padding:0px; }
--></style>
<base target="inhalt"></head>
<body>
<p>
<a href="home.html">HOME</a> |
<a href="buch.html">BUCH</a> |
<a href="forum.html">FORUM</a> |
<a href="kontakt.html">KONTAKT</a> |
<a href="mehr.html">MEHR SCRIPTS</a>
</p></body></html>
```

Listing 16.9: Die Datei menu.html

```
<html><head>
<title></title>
<link rel="stylesheet" href="seiten.css"
type="text/css">
</head>
<body></body>
</html>
```

Listing 16.10: Die Datei blind.html

Die Datei menu.html

Diese Datei gestaltet das Menü nach Ihren Wünschen. Im Beispiel wurde ein einfaches Textmenü gewählt (siehe Listing 16.9).

Die Datei blind.html

Wenn das Navigationsmenü ausgeblendet werden soll, wird die Datei *ohne_menu.html* aufgerufen. In der Datei ist der Frame `menu` auf `1px` gesetzt; damit wird er quasi unsichtbar.

Damit keine Fehlermeldung erscheint, wird in diesem Frame die leere Datei *blind.html* aufgerufen. Sie verfügt außer dem HTML-Grundgerüst über keine weiteren Inhalte. Lediglich die Hintergrundfarbe sollte auf die angrenzende Inhaltsseite angepasst werden. Im Beispiel wird das durch die Angaben in der CSS-Datei *seiten.css* erledigt (siehe Listing 16.10).

Die Datei lade.html

Angenommen, der Besucher befindet sich auf der Unterseite *Leseprobe* in der Rubrik *Buch* und klappt das Menü zu. Die Datei *lade.html* sorgt beim Aus- und Einblenden des Menü-Frames dafür, dass die Seite Leseprobe wieder aufgerufen wird. Diese Aufgabe erledigt das Script `LadeSeite()` in der Datei *lade.html*.

1. `var Seite;`
   Zuerst wird die globale Variable `Seite` erstellt. Sie steht für die aktuelle Seite, die gerade im Frame `inhalt` aufgerufen ist.
2. `Seite = top.steuerung.anzeigen;`
   Im Frame `steuerung` ist das Script zum Aus- und Einblenden des Menüs untergebracht. In diesem Script wird die Seite, die aktuell im Frame `inhalt` zu sehen ist in der Variablen `anzeigen` gespeichert. Mit `top.steuerung.anzeigen` wird der Wert dieser Variablen angesprochen und an die Variable `Seite` der Funktion `LadeSeite` übergeben.
3. `if (Seite == null)`
   Wenn kein Wert vorhanden ist (z.B. beim Aufruf der Website), wird die Datei *home.html* geladen. Das ist die eigentliche Startseite der Website, auf der die Besucher begrüßt werden. Passen Sie den Dateinamen also bei Bedarf an.
4. `else`
   Ist in der Variablen ein Wert vorhanden, wird mit der `else`-Anweisung die aktuelle Seite geladen.

Listing 16.11 zeigt den Quellcode der Datei *lade.html*.

```
<html><head>
<title>Dynamische Frames</title>
<script language="JavaScript">
<!--
var Seite;

function LadeSeite() {
Seite = top.steuerung.anzeigen;
if (Seite == null) {
self.location = 'home.html';
}
else {
self.location = Seite;
}
}

//-->
</script></head>
<body onLoad="LadeSeite()">
Moment bitte, Seite wird geladen...
</body></html>
```

Listing 16.11: Die Datei lade.html

✓ Tipp:

- Für die Datei *lade.html* ist im `<body>`-Tag eigentlich kein Inhalt erforderlich. Sie sollten dem Besucher aber einen Hinweis geben, dass die Seite geladen wird, denn sonst sieht er nur eine leere Seite, wenn das Laden mal etwas länger dauert.

```
<html><head>
<title>Schalter</title>
<style type="text/css">
<!--
body { margin:0px 0px 0px 0px; background-
color:#666666; }
a { font:bold 12pt Verdana,Arial,Helve-
tica,sans-serif; color:#F0F0F0; text-deco-
ration:none; }
--></style>
</head>
<body>
<a href="#" onClick="top.steuerung.Steu-
ern(); this.blur();">
<img src="../einaus.gif" width="24"
height="150" border="0" alt="Menu ein-
oder ausblenden"></a>
</body></html>
```

Listing 16.12: Die Datei schalten.html

Die Datei schalten.html

Die Datei *schalten.html* wird im Frameset *index.html* aufgerufen. Sie enthält den Link, der den Menü-Frame aus- und einblendet.

1. `onClick="top.steuerung.Steuern();`
   Das Script `Steuern()` ist in der Datei *steuerung.html* notiert; diese wird im Frame `steuerung` aufgerufen. Der Zugriff auf das Script erfolgt mit dem Event-Handler `onClick` über `top.steuerung.Steuern();`.
2. `this.blur();`
   Im Internet Explorer wird um angeklickte Links eine dünne gestrichelte Linie gezogen. Sie wird durch `this.blur();` entfernt.

Den Quelltext der Datei *schalten.html* finden Sie in Listing 16.12.

Die Datei steuerung.html

Nun kommen wir zum Script, das die Steuerung des Effekts übernimmt. Es befindet sich in der Datei *steuerung.html* im Frame `steuerung`.

1. `var EinAus = 0;`
   `var anzeigen = null;`
   Die Variable `EinAus` speichert den Status des Framesets und wird mit `0` initialisiert. `anzeigen` speichert die Seite, die aktuell im Frame `inhalt` aufgerufen ist.
2. `if(EinAus == 0)`
   Beim Laden der Website ist der Menü-Frame zu sehen, der Status von `EinAus` ist `0`. Beim Aufruf der Funktion `Steuern()` wird in diesem Zustand im Frame `haupt` über `top.haupt.location` das Frameset *ohne_menu.html* aufgerufen und dann die Variable `EinAus auf 1` gesetzt.
3. `else`
   Die `else`-Anweisung reagiert, wenn `EinAus` auf `1` gesetzt ist, der Menü-Frame also ausgeblendet ist. In diesem Fall wird beim Aufruf von `Steuern()` im Frame `haupt` das Frameset *mit_menu.html* aufgerufen und der Wert von `EinAus` wieder auf `0` gesetzt.
4. `parent.haupt.inhalt.location.href`
   Es handelt sich um ein verschachteltes Frameset, der Frame `inhalt` ist ein Element des Frames `haupt`. Der Rahmen `inhalt` muss daher über `parent.haupt.inhalt` angesprochen werden. Die Seite, die aktuell in `inhalt` aufgerufen ist, wird in der Variablen `anzeigen` gespeichert.

Sie finden den Quelltext in Listing 16.13.

```
<html><head>
<title>Dynamische Frames</title>
<script language="JavaScript">
<!--
var EinAus = 0;
var anzeigen = null;

function Steuern() {
//mit Menu
if(EinAus == 0) {
anzeigen = parent.haupt.inhalt.
location.href;
top.haupt.location = 'ohne_menu.html';
EinAus = 1;
return;
}
//ohne Menu
else {
anzeigen = parent.haupt.inhalt.
location.href;
top.haupt.location= 'mit_menu.html';
EinAus = 0;
return;
}
}
//-->
</script></head>
<body></body></html>
```

Listing 16.13: Die Datei steuerung.html

✓ Tipp:

- Im Beispiel ist der Frame `steuerung` nur 1px breit und dadurch in der Webseite nicht zu sehen. Die Datei enthält daher außer dem Script auch keine weiteren Inhalte. Der Frame kann aber auch dazu dienen, Informationen aufzunehmen, die immer sichtbar sein sollen, z.B. ein Copyright-Vermerk oder die E-Mail-Adresse des Webmasters.

Abbildung 16.14: Die Seite nach dem Laden. Wenn man den Link Buch anklickt…

Abbildung 16.15: …erscheint die entsprechende Inhaltsseite.

Abbildung 16.16: Die Seite ist auch noch zu sehen, nachdem der Link angeklickt wurde, der den Menüframe ausblendet.

Dynamische Frames in Aktion

In Abbildung 16.14 sehen Sie die Beispielseite nach dem Laden. Der Menüframe ist zu sehen, im Frame `inhalt` wurde die Homepage (`home.html`) aufgerufen.

Der Link `Buch` wird angeklickt, im Frame `inhalt` erscheint die entsprechende Seite (siehe Abbildung 16.15).

Wenn man den Aus-/Ein-Button anklickt, wird das Menü ausgeblendet und die aktuell aufgerufene Seite wird neu geladen (siehe Abbildung 16.16).

Dropdown-Menüs

Dropdown-Menüs benutzen Sie am Computer sehr oft, denn fast jedes Programm verwendet solche Menüs (siehe Abbildung 16.17).

Einfache Dropdown-Menüs sind über Ebenen, die auf Mausklick ein- und ausgeblendet werden, für Webseiten leicht zu realisieren (siehe Abbildung 16.18).

Wenn man eine Rubrik anklickt, wird ein bereits geöffnetes Menü geschlossen und stattdessen das Menü der aktuellen Rubrik eingeblendet. Dazu braucht man lediglich die Funktionen `f8()` (zum Einblenden) und `f9()` (zum Ausblenden), die in Kapitel 13 besprochen wurden.

Mit Hilfe eines Grafikprogramms und durch Hinzufügen des Rollover-Effekts lassen sich auch komplexere Menüstrukturen realisieren, die den Software-Menüs täuschend ähnlich sehen. Außer den Funktionen Einblenden, Ausblenden und Rollover sind nur etwas Geschick im Umgang mit der Bildbearbeitung und einige Überlegungen erforderlich, um ein solches Menü zu realisieren.

In Abbildung 16.19 sehen Sie den Entwurf für das Menü meiner Website. Das komplette Menü finden Sie als Workshop auf der Support-Seite zum Buch; dort wird auch erklärt, wie das Menü aufgebaut wurde. Die PhotoShop-Datei, die als Grundlage für das Menü diente, steht ebenfalls zum Download zur Verfügung.

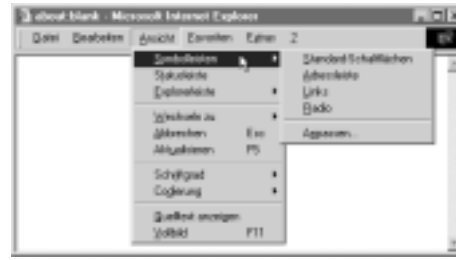

Abbildung 16.17: Fast jede Software nutzt Dropdown-Menüs.

Abbildung 16.18: Das Dropdown-Menü der Rubrik Buch

Abbildung 16.19: Auch komplexere Menüs sind möglich.

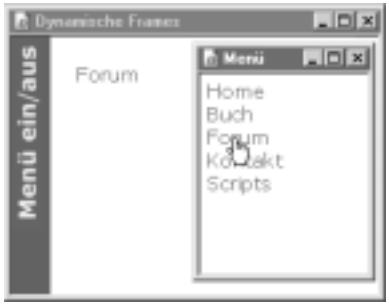

Abbildung 16.20: Über das Zusatzfenster…

Abbildung 16.21: …werden die Inhalte des Hauptfensters gesteuert.

Fernsteuerung für die Website

Sie können dem Besucher die Möglichkeit bieten, die Seitennavigation in einem zweiten Fenster aufzurufen und die Inhalte des Hauptfensters vom Zusatzfenster aus zu steuern, siehe Abbildung 16.20 und Abbildung 16.21.

Das kommt vor allem Besuchern mit kleinen Monitoren zugute.

Da in einigen Browsern (z.B. Opera) die Möglichkeit besteht, Popup-Fenster trotz eingeschalteter JavaScript-Unterstützung zu unterdrücken, sollten Sie in jedem Fall beim Starten der Seite eine alternative Navigation anzeigen und dem Besucher die Wahl lassen.

Ebenso sollte der Anwender jederzeit die Möglichkeit haben, von der Fernsteuerung auf die normale Navigation umzuschalten.

In Abbildung 16.20 und Abbildung 16.21 wurde die Fernsteuerung zusammen mit den dynamischen Frames verwendet, die bereits besprochen wurden.

Die Funktion zur Fernsteuerung

Wie man ein neues Browserfenster öffnet und schließt, wurde bereits in Kapitel 14 behandelt. In Listing 16.14 sehen Sie die beiden Funktionen noch einmal.

Damit man im Hauptfenster nur einen Link benötigt, wurden die beiden Funktionen zusammengefasst. Das neue Script wird in einer externen Datei gespeichert; im Beispiel heißt sie *fern.js*.

1. `fern=null;`
   Für das neue Fenster wird die globale Variable `fern` mit `null` initialisiert.
2. `if(fern==null)`
   Wenn das neue Fenster noch nicht geöffnet wurde, ist der Wert der Variable `fern` gleich `null`. In diesem Fall wird das neue Fenster mit `window.open` unter Angabe der Fensterparameter geöffnet.
3. `else`
   Wenn das Fenster bereits offen ist, tritt die `else`-Anweisung in Funktion; das Fenster wird geschlossen.
4. `//document.write`
   Wenn die Fernsteuerung nicht von einem Frameset aus aufgerufen wird, entfernen Sie die Kommentarzeichen // vor dem Befehl `document.write`. Damit kann außerhalb der Funktion der Link in die Seite geschrieben werden und man muss dies nicht auf jeder Seite wiederholen. Beachten Sie, dass die Anführungszeichen hier maskiert werden müssen. Dazu wird der Backslash \ vorangestellt.

Den Quelltext der Datei *fern.js* sehen Sie in Listing 16.15.

```
function e5(Datei,FName,Aussehen) {
NeuesFenster = win-
dow.open(Datei,FName,Aussehen);
}

function e8() {
window.close()
}
```

Listing 16.14: Die Funktionen zum Öffnen und Schließen eines neuen Fensters

```
//Inhalt der Datei fern.js:
fern=null;

function fernEinAus() {
if(fern==null) {
fern=window.open('menu.html','fern','resiz
able=yes,width=150,height=200');
fern.focus();
}
else {
fern.close();
fern = null;
}
}

//document.write("<a href=\"#\"
onClick=\"fernEinAus()\"> Fernsteuerung
ein/aus</a>");
```

Listing 16.15: Die Funktionen zum Öffnen und Schließen eines neuen Fensters. Im Beispiel ist `document.write` auskommentiert. Der Befehl kommt zum Einsatz, wenn die Fernsteuerung nicht von einem Frameset, sondern von einzelnen Seiten aus aufgerufen wird.

```
<html><head>
<title>Titel der Seite</title>
</head>
<link rel="stylesheet" href="seiten.css"
type="text/css">
<body>
<p>
<script src="fern.js"></script>
</p>
<p>Andere Inhalte…</p>
</body></html>
```

Listing 16.16: Der Quelltext einer Inhaltsseite

```
<html><head>
<title>Fernsteuerung</title>
<link rel="stylesheet" href="seiten.css"
type="text/css"></head>
<body onUnload="opener.fern = null">
<p><a href="seite1.html"
target="FrameName">Seite 1</a></p>
<p><a href="seite2.html"
target="FrameName">Seite 2</a></p>
</body></html>
```

Listing 16.17: Die Seite für das Zusatzfenster, wenn es von einem Frameset aus aufgerufen wird

Aufruf der Funktion

In den Inhaltsseiten wird die Datei *fern.js* eingebunden. Wenn Sie den `document.write`-Befehl aktiviert haben, wird das Script nicht im Head-Bereich eingebunden, sondern im `<body>` an der Stelle, an der der Link zum Aufruf der Fernsteuerung stehen soll (siehe Listing 16.16).

Die Seite für die Fernsteuerung

Im `<body>`-Tag der Seite, die im Zusatzfenster aufgerufen wird, muss der Event-Handler `onUnload` gesetzt werden. Als Wert wird `opener.fern = null` angegeben. Mit dem Fensterobjekt `opener` hat man Zugriff auf das Fenster, von dem aus die Fernsteuerung geöffnet wurde. Hier wird die Variable `fern` wieder auf `null` gesetzt (siehe Listing 16.17).

Einsatz mit Frameset

Wie kann man auf die Inhalte des Hauptfensters zugreifen? Wenn die Fernsteuerung von einem Frameset aus aufgerufen wird, reicht es, bei den Links mit dem Attribut `target` den gewünschten Frame anzugeben (siehe Listing 16.17).

Einsatz ohne Frameset

Wenn die Fernsteuerung nicht von einem Frameset, sondern von einzelnen Seiten aus aufgerufen wird, erfolgt der Zugriff auf das Hauptfenster anders als beim Frameset.

Man nimmt den Weg über das Objekt `opener`, mit dem man Zugriff auf das Fenster hat, von dem aus das Zusatzfenster geöffnet wird. In der Datei, die als Fernsteuerung dient, wird die Funktion `ziel` ergänzt. Die neue Funktion wird in den Links aufgerufen.

1.
```
function ziel(Seite) {
Seite;
opener.location.href=Seite;
}
```
Für die Seiten, die im Hauptfenster aufgerufen werden, wird die Variable `Seite` gesetzt. Der Befehl `opener.location.href` spricht das Hauptfenster an.

2. `onClick="ziel('seite1.html')"`
In den Links wird mit dem Event-Handler `onClick` die Funktion `ziel` aufgerufen. Als Parameter wird die URL der Seite angegeben, die im Hauptfenster aufgerufen werden soll.

Den Quelltext für die Seite, die als Fernsteuerung dient, sehen Sie in Listing 16.18. Im Head wurde die Funktion `ziel` hinzugefügt.

```
<html><head>
<title>Menü</title>
<script language="JavaScript">
<!--
function ziel(Seite) {
Seite;
opener.location.href = Seite;
}
//-->
</script></head>
<body onUnload="opener.fern = null">
<p><a href="#"
onClick="ziel('seite1.html')">
Seite 1</a></p>
<p><a href="#"
onClick="ziel('seite2.html')">
Seite 2</a></p>
</body></html>
```

Listing 16.18: Die Fernsteuerung, wenn das Fenster von einer einzelnen Seite aus aufgerufen wird

✓ Tipp:

- Man könnte in der Datei *fern.js* mit `window.name="HauptfensterName"` für das Hauptfenster einen Namen vergeben und in der Fernsteuerung bei den Links als `target`-Wert dann den vergebenen Namen des Hauptfensters setzen. Opera öffnet den ersten angeklickten Link aber in einem neuen Fenster statt im Hauptfenster. Alle weiteren Links werden zwar ebenfalls in dem neuen Fenster geöffnet, die Fernsteuerung funktioniert also ab dem zweiten Link normal, trotzdem wirkt das unprofessionell.

```
<html><head>
<title>Dynamische Frames</title>
<script language="JavaScript">
<!--
var EinAus = 0;
var anzeigen = null;
fern=null;
function fernEinAus() {
if(fern==null) {
fern=window.open('fernmenu.html',
'fern','resiz-
able=yes,width=150,height=200');
fern.focus();
}
else {
fern.close();
fern = null;
}
}

function Steuern() {
//mit Menu
if(EinAus == 0) {
fernEinAus();
anzeigen = parent.haupt.inhalt.loca-
tion.href;
top.haupt.location='ohne_menu.html';
EinAus = 1;
return;
}
//ohne Menu
else {
fernEinAus();
anzeigen = parent.haupt.inhalt.loca-
tion.href;
top.haupt.location='mit_menu.html';
EinAus = 0;
return;
}
}
//-->
</script></head>
<body></body></html>
```

Listing 16.19: Die ergänzte Datei steuerung.html für die dynamischen Frames

Einsatz mit dynamischen Frames

Zu den dynamischen Frames (siehe den Abschnitt *Dynamische Frames* in diesem Kapitel), ist eine Fernsteuerung die optimale Ergänzung.

Damit kein zweiter Link benötigt wird, kann man die Datei *steuerung.html* so modifizieren, dass der Link, mit dem der Menüframe zugeklappt wird, gleichzeitig auch die Fernsteuerung aufruft oder schließt.

Dazu wird im Head der Datei *steuerung.html* die Funktion zum Öffnen und Schließen des Fensters ergänzt. In der Funktion, die für die Menü-Steuerung zuständig ist, wird in beiden Verzweigungen die Fensterfunktion aufgerufen (siehe Listing 16.19).

In Aktion sehen Sie die Kombination der beiden Navigationen auf der nächsten Seite.

Schritt 1

Abbildung 16.22: Nach dem Aufruf der Website wurde der Link Forum angeklickt. Diese Seite wird im Inhaltsframe aufgerufen. Der Klick auf den Menüschalter…

Schritt 2

Abbildung 16.23: … klappt das Menü zu und ruft gleichzeitig die Fernsteuerung auf. Dort wird nun der Link Kontakt gewählt…

Schritt 3

Abbildung 16.24: …und somit die entsprechende Seite im Inhaltsframe aufgerufen. Wenn man das Menü im Frameset mit einem Klick auf den Schalter wieder aufklappt…

Schritt 4

Abbildung 16.25: …wird die Fernsteuerung geschlossen., Das Menü ist aufgeklappt und im Inhaltsframe steht weiterhin die Seite, die gerade aufgerufen war.

Abbildung 16.26: Die Internetseite zum Buch

Die Website zum Buch

Die Grundlagen für DHTML kennen Sie nun. Bauen Sie darauf auf und verwirklichen Sie Ihre eigenen Ideen.

Sparen Sie sich die Mühe, die Scripts abzutippen. Sie finden alle Listings auf der Website zum Buch unter der Internetadresse *http://www.css-dhtml.de*

Auf der Internetseite stehen Ihnen neben den Listings aus dem Buch weitere Angebote zur Verfügung, z.B. die CSS-Referenz und ein Forum, in dem Sie Fragen stellen können.

Einige Bereiche der Website sind nur Ihnen, den Leserinnen und Lesern des Buchs, zugänglich und daher passwortgeschützt. Bitte verwenden Sie für diese Bereiche die folgenden Login-Daten:

Login-Name: `Leserseiten`

Passwort: `C2K6Leser`

Surfen Sie doch einfach mal vorbei – ein Besuch lohnt sich bestimmt.

A

NAMENSKONVENTIONEN

Wenn Sie mit CSS und DHTML arbeiten, müssen Sie Namen für Klassen, IDs, Variablen und Funktionen vergeben.

Auf den nachfolgenden Seiten ist noch einmal zusammengefasst, was Sie bei der Vergabe der Namen beachten müssen. Sie finden auch eine Liste der Wörter, die Sie für Namen nicht verwenden können, weil sie für JavaScript reserviert sind oder in CSS und JavaScript eine besondere Bedeutung haben.

Namen vergeben

Folgende Konventionen müssen Sie bei der Vergabe von Namen für Klassen, IDs, Variablen und Funktionen beachten.

- Verwenden Sie nur die Buchstaben aA-zZ und Ziffern, keine Sonderzeichen oder Umlaute.
- Die Namen müssen mit einem Buchstaben beginnen.
- Wörter, die für JavaScript und Java reserviert sind, dürfen für Namen nicht verwendet werden.
- Auch Begriffe, die in HTML eine besondere Bedeutung haben, müssen Sie vermeiden. Dazu gehören die HTML-Selektoren, z.B. `table`, aber auch Attributnamen wie `checked`.
- Gleiches gilt für Eigenschaften und Methoden aus JavaScript und CSS sowie einige andere Begriffe.

Eine alphabetisch sortierte Liste mit den Wörtern, die Sie nicht für Namen verwenden sollten, finden Sie den nachfolgenden Seiten.

✓ Tipps:

- In den nachfolgenden Listen sind viele Begriffe nur in Kleinschreibung aufgeführt. Dennoch sollten Sie in jedem Fall auch die großgeschriebenen Begriffe vermeiden. Wenn in der Liste das Wort `status` aufgeführt ist, sollten Sie also auch den Begriff `Status` nicht verwenden. Umgekehrt gilt das Gleiche.
- Eine vollständige Liste der HTML-Selektoren und Attributnamen finden Sie z.B. auf der Seite des W3C unter *http://www.w3.org*
- Wenn ein Script nicht läuft, prüfen Sie auch die Schreibweise des Namens und benennen Sie die Variablen und/oder Funktion um.

Die nachfolgenden Wörter sollten Sie für Namen nicht verwenden:

above	availWidth	className
abs	back	clearInterval
abstract	background	clearTimeout
acos	below	click
action	bgColor	clientX
alert	big	clientY
alinkColor	blink	clip
all	blur	cloneNode
all	bold	close
altKey	boolean	closed
anchor	border	colorDepth
anchors	break	complete
appCodeName	byte	concat
appendChild	caller	confirm
appendData	captureEvents	const
applets	case	contains
appName	catch	continue
appVersion	ceil	cookie
arguments	char	cookieEnabled
arity	charAt	cos
Array	charCodeAt	createAttribute
asin	charset	createElement
atan	checked	createTextNode
attributes	childNotes	ctrlKey
availHeight	class	data

dataFld	encoding	fromCharCode
dataFormatAs	event	function
dataPageSize	exec	getAttribute
dataSrc	exp	getAttributeNode
Date	export	getDate
default	extends	getDay
default-Charset	External Capture	getElementById
default-Checked	false	getElementsByName
default-Selected	fgColor	getElementsByTagName
defaultStatus	filename	getFullYear
defaultValue	final	getHours
delete	finally	getMilliseconds
deleteData	find	getMinutes
description	firstChild	getMonth
disable	fixed	getSeconds
do	float	getSelection
document	floor	getTime
double	focus	getTimezoneOffset
E	fontcolor	getUTCDate
element	fontsize	getUTCDay
elements	for	getUTCFullYear
else	form	getUTCHours
embeds	forms	getUTCMilliseconds
enable	forward	getUTCMinutes
enabledPlugin	frames	getUTCMonth

Fortsetzung auf der nächsten Seite

Fortsetzung

getUTCSeconds	insertAdjacentHTML	LN10
getYear	insertAdjacentText	LN2
go	insertBefore	load
goto	insertData	location
handleEvent	instanceof	locationbar
hasChildNodes	int	log
hash	isTextEdit	LOG10E
height	italic	LOG2E
height	italics	long
history	javaEnabled	lowsrc
host	join	match
hostname	keyCode	Math
href	lang	max
hspace	language	MAX_VALUE
id	lastChild	menubar
if	lastIndexOf	method
images	lastModified	mimeTypes
implements	layers	min
import	layerX	MIN_VALUE
in	layerY	modifiers
indexOf	left	moveAbove
innerHeight	length	moveBelow
innerHTML	link	moveBy
innerText	linkColor	moveTo
innerWidth	links	name

NaN	pageX	public
native	pageXOffset	push
navigator	pageY	random
new	pageYOffset	recordNumber
nextSiblin g	parentEleme nt	referrer
node	parentLayer	RegExp
nodeName	parentNode	releaseEvents
nodeType	parentText- Edit	reload
nodeValue	parse	removeAttribute
null	pathname	removeAttribute- Node
Number	personalbar	removeChild
offset- Height	PI	replace
offsetLeft	pixelDepth	replaceChild
offsetPare nt	platform	replaceData
offsetTop	play	reset
offset- Width	plugins	resizeBy
offsetX	pop	resizeTo
offsetY	port	return
open	pow	reverse
options	previous- Sibling	round
outer- Height	print	routeEvent
outer- HTML	private	Screen
outerText	prompt	screenX
outer- Width	protected	screenY
package	protocol	scrollbars

Fortsetzung auf der nächsten Seite

Fortsetzung

scrollBy	setUTCSeconds	sub
scrollIntoView	setYear	submit
scrollTo	shift	substr
search	shiftKey	substring
select	short	suffixes
selected	siblingAbove	sup
selectedIndex	siblingBelow	super
setAttribute	sin	switch
setAttributeNode	slice	synchronized
setDate	small	tagName
setFullYear	sort	tan
setHours	sourceIndex	target
setInterval	splice	test
setMilliseconds	split	text
setMinutes	sqrt	this
setSeconds	SQRT1_2	throw
setTime	SQRT2	throws
setTimeout	src	title
setUTCDate	static	toExponential
setUTCDay	status	toFixed
setUTCFullYear	statusbar	toGMTString
setUTCHours	stop	toLocaleStrin g
setUTCMilliseconds	strike	toLowerCase
setUTCMinutes	string	toolbar
setUTCMonth	style	top

toPrecision	x	
toString	y	
toUpperCase	zIndex	
transient		
true		
try		
type		
typeof		
unshift		
URL		
userAgent		
UTC		
value		
var		
visibility		
vlinkColor		
void		
vspace		
which		
while		
width		
window		
with		
write		
writeln		

B

CSS-REFERENZ

In der CSS-Referenz finden Sie alle im Buch besprochenen CSS-Eigenschaften mit den erlaubten Attributen. Die Eigenschaften sind in der Reihenfolge der Kapitel aufgeführt.

Nähere Informationen zu den Tabellen finden Sie auf der nächsten Seite.

Die Referenztabellen

Die Tabellen auf den nachfolgenden Seiten sind thematisch nach Kapiteln geordnet. Sie beinhalten die Eigenschaften mit den dazugehörigen erlaubten Werten. Hier können Sie nachschlagen, welcher Browser eine bestimmte Eigenschaft bzw. Wert unterstützt. Innerhalb der Tabellen sind die Eigenschaften alphabetisch sortiert.

Außerdem ist in den Tabellen angegeben, ob eine Eigenschaft an Kindelemente vererbt wird und auf welche Tag-Gruppe sie angewendet werden kann.

Auf der Webseite zum Buch finden Sie die Referenz ebenfalls; dort wird sie bei Bedarf aktualisiert.

Beachten Sie die nebenstehende Legende zum Lesen der Tabellen.

Legende zur Referenz

In den nachfolgenden Tabellen werden die folgenden Abkürzungen für Browser verwendet:

- IE4 = Internet Explorer 4
- IE5 = Internet Explorer 5
- N4 = Netscape 4
- N6 = Netscape 6
- O5 = Opera 5
- O6 = Opera 6

Browserunterstützung:

- J = Ja
- N = Nein
- B = Buggy; es können (müssen aber nicht) Probleme auftreten

T: Anwendung in Tag-Gruppe:

- A = Alle Tags
- B = Blocklevel-Tags
- I = Inline-Tags

V: Vererbung der Eigenschaft

- J = Ja
- N = Nein

Standardwerte sind hervorgehoben. Beachten Sie auch ggf. besondere Kennzeichnungen für bestimmte Eigenschaften oder Werte.

Kapitel 3: Schriftformatierung				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
color	<Farbangabe>	J	A	J	J	J	J	J	J	J	J	J	J
font-family	<Schriftarten>	J	A	J	J	J	J	J	J	J	J	J	J
	sans-serif			J	J	J	J	J	J	J	J	J	J
	serif			J	J	J	J	J	J	J	J	J	J
	monospace			J	J	J	J	J	J	J	J	J	J
	cursive			N	J	J	J	J	J	J	J	J	J
	fantasy			N	J	J	J	J	J	J	J	J	J
font-size	<Maße>	J	A	J	J	J	J	J	J	J	J	J	J
	<Prozentangaben>			J	J	J	J	J	J	J	J	J	J
	smaller			J	J	B	J	J	J	J	J	B	B
	larger			J	J	B	J	J	J	J	J	B	B
	xx-small			J	J	B	J	B	J	J	J	B	B
	x-small			J	J	B	J	B	J	J	J	B	B
	small			J	J	B	J	B	J	J	J	B	B
	medium			J	J	B	J	B	J	J	J	B	B
	large			J	J	B	J	B	J	J	J	B	B
	x-large			J	J	B	J	B	J	J	J	B	B
	xx-large			J	J	B	J	B	J	J	J	B	B
font-style	normal	J	A	J	J	J	J	J	J	J	J	J	J
	italic			J	J	J	J	J	J	J	J	J	J
	oblique			N	J	J	J	J	J	N	J	J	J
font-variant	normal	J	A	N	J	J	J	J	J	N	J	J	J
	small-caps			N	J	B	J	J	J	N	J	B	J
font-weight	normal	J	A	J	J	J	J	J	J	J	J	J	J
	bold			J	J	J	J	J	J	J	J	J	J
	bolder			J	J	J	J	J	J	N	J	J	J
	lighter			N	J	J	J	J	J	N	J	J	J
	100-900*			J	J	J	J	J	J	J	J	J	J
line-height	normal	J	A	J	J	J	J	J	J	J	J	J	J
	<numerische Werte>			B	J	J	J	J	J	B	J	J	J
font	<font-style>	J	A	J	J	J	J	J	J	J	J	J	J
	<font-variant>			N	J	B	J	J	J	N	J	B	J
	<font-weight>			J	J	J	J	J	J	J	J	J	J
	<font-size>/			J	J	B	J	B	J	J	J	B	B
	<line-height>			J	J	J	J	J	J	J	J	J	J
	<font-family>			J	J	J	J	J	J	J	J	J	J

*Abhängig von der Schriftart, die auf dem Besucherrechner installiert ist

Kapitel 4: Textformatierung				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
letter-spacing	normal	J	A	N	J	J	J	J	J	N	J	J	J
	<Maße>			N	J	J	J	J	J	N	J	J	J
text-align	left	J	B	J	J	J	J	J	J	J	J	J	J
	right			J	J	J	J	J	J	J	J	J	J
	center			J	J	J	J	J	J	J	J	J	J
	justify			B	J	J	J	J	J	B	J	N	J
text-decoration	none	N	A	J	J	J	J	J	J	J	J	J	J
	blink			J	J	N	N	J	J	J	J	N	N
	line-through			J	J	J	J	J	J	J	J	J	J
	overline			N	J	J	J	J	J	N	J	J	J
	underline			J	J	J	J	J	J	J	J	J	J
text-indent	<Maße>	J	B	J	J	J	J	J	J	J	J	J	J
	<Prozentangaben>			J	J	J	J	J	J	J	J	J	J
text-transform	none	J	A	J	J	J	J	J	J	J	J	J	J
	capitalize			J	J	J	J	J	J	J	J	J	J
	lowercase			J	J	J	J	J	J	J	J	J	J
	uppercase			J	J	J	J	J	J	J	J	J	J
vertical-align	baseline	N	I	N	J	J	J	J	J	N	J	J	J
	top			N	J	N	J*	J	J	N	J	J	J
	text-top			N	J	N	J*	J	J	N	J	J	J
	middle			N	J	N	J*	J	J	N	J	J	J
	bottom			N	J	N	J*	J	J	N	J	B	J
	text-bottom			N	J	N	J*	J	J	N	J	B	J
	sub			N	J	J	J	J	J	N	J	J	J
	super			N	J	J	J	J	J	N	J	J	J
	<Prozentangaben>			N	J	N	N*	J	J	N	J	B	J
white-space	normal	J	B	J	J	N	J*	J	J	J	J	N	J
	nowrap			N	J	N	J*	J	J	N	J	N	J
	pre			J	J	N	N	J	J	J	J	N	J
word-spacing	<Maße>	J	A	N	J	N	N	J	J	N	J	J	J
	normal			N	J	N	N	J	J	N	J	J	J

*Nur IE5.5, nicht in IE5.0

Kapitel 5: Listen und Maus-Cursor				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
cursor	auto	J	A	N	J	J	J	N	N	N	J	J	J
	crosshair			N	J	J	J	N	N	N	J	J	J
	hand			N	N	J	J	N	N	N	N	J	J
	help			N	J	J	J	N	N	N	J	J	J
	move			N	J	J	J	N	N	N	J	J	J
	pointer			N	J	N	N	N	N	N	J	N	N
	text			N	J	J	J	N	N	N	J	J	J
	wait			N	J	J	J	N	N	N	J	J	J
	e-resize			N	J	J	J	N	N	N	J	J	J
	ne-resize			N	J	J	J	N	N	N	J	J	J
	n-resize			N	J	J	J	N	N	N	J	J	J
	nw-resize			N	J	J	J	N	N	N	J	J	J
	w-resize			N	J	J	J	N	N	N	J	J	J
	sw-resize			N	J	J	J	N	N	N	J	J	J
	s-resize			N	J	J	J	N	N	N	J	J	J
	se-resize			N	J	J	J	N	N	N	J	J	J
list-style-image	none	J	A	N	J	J	J	J	J	N	J	J	J
	url(<url>)			N	J	J	J	J	J	N	J	J	J
list-style-position	outside	J	A	N	J	J	J	J	J	N	J	B	J
	inside			N	J	J	J	J	J	N	J	J	J
list-style-type	disc	J	A	J	J	J	J	J	J	J	J	J	J
	circle			J	J	J	J	J	J	J	J	J	J
	square			J	J	J	J	J	J	J	J	J	J
	decimal			J	J	J	J	J	J	J	J	J	J
	lower-roman			J	J	J	J	J	J	J	J	J	J
	upper-roman			J	J	J	J	J	J	J	J	J	J
	lower-alpha			J	J	J	J	J	J	J	J	J	J
	upper-alpha			J	J	J	J	J	J	J	J	J	J
	none			J	J	J	J	J	J	J	J	J	J
list-style	list-style-image	J	A	N	J	J	J	J	J	N	J	J	J
	list-style-position			N	J	J	J	J	J	N	J	B	J
	list-style-type			J	J	J	J	J	J	J	J	J	J

Kapitel 5: Pseudo-Formate				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
a:link	---	J	<a>	J	J	J	J	J	J	J	J	J	J
a:visited	---	J	<a>	N	J	J	J	J	J	N	J	J	J
a:hover	---	J	<a>	N	J	J	J	J	J	N	J	J	J
a:active	---	J	<a>	N	J	J	J	J	J	N	J	J	J
:first-letter	---	N	B	N	J	N	J*	J	J	N	J	N	J
:first-line	---	N	B	N	J	N	J*	J	J	N	J	N	J

* Nur IE5.5, nicht in IE5.0

Kapitel 6: Hintergründe				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
background-attachment	scroll	N	A	N	J	J	J	J	J	N	J	J	J
	fixed												
background-color	transparent	N	A	J	J	J	J	J	J	J	J	J	J
	<Farbangabe>												
background-image	none	N	A	B	J	J	J	J	J	B	J	J	J
	url(<url>)												
background-repeat	repeat	N	A	J	J	B	J	J	J	J	J	J	J
	repeat-x			B	J	B	J	J	J	B	J	J	J
	repeat-y			B	J	B	J	J	J	B	J	J	J
	no-repeat			J	J	J	J	J	J	J	J	J	J
background-position	top	N	B	N	J	J	J	J	J	N	J	J	J
	center			N	J	J	J	J	J	N	J	J	J
	bottom			N	J	J	J	J	J	N	J	J	J
	left			N	J	J	J	J	J	N	J	J	J
	right			N	J	J	J	J	J	N	J	J	J
	<Maße>			N	J	J	J	J	J	N	J	J	J
	<Prozentangaben>			N	J	J	J	J	J	N	J	J	J
background	background-attachment	N	A	N	J	J	J	J	J	N	J	J	J
	background-color			J	J	J	J	J	J	J	J	J	J
	background-image			J	J	J	J	J	J	J	J	J	J
	background-repeat			J	J	J	J	J	J	J	J	J	J
	background-position			N	J	J	J	J	J	N	J	J	J

Kapitel 7: Rahmeneigenschaften				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
border-color	<Farbangaben>	N	A	B	J	J	J	J	J	B	J	J	J
border-top-color				B	J	J	J	J	J	B	J	J	J
border-right-color				B	J	J	J	J	J	B	J	J	J
border-bottom-color				B	J	J	J	J	J	B	J	J	J
border-left-color				B	J	J	J	J	J	B	J	J	J
border-style	none	N	A	J	J	J	J	J	J	J	J	J	J
border-top-style	dashed			N	J	N	J*	J	J	N	J	J	J
border-right-style	dotted			N	J	N	J*	J	J	N	J	J	J
border-bottom-style	double			J	J	J	J	J	J	J	J	J	J
border-left-style	groove			J	J	J	J	J	J	J	J	J	J
	inset			J	J	J	J	J	J	J	J	J	J
	outset			J	J	J	J	J	J	J	J	J	J
	ridge			J	J	J	J	J	J	J	J	J	J
	solid			J	J	J	J	J	J	J	J	J	J
border-width	medium	N	A	J	J	B	J*	J	J	J	J	B	J
border-top-width	thick			J	J	B	J*	J	J	J	J	B	J
border-right-width	thin			J	J	B	J*	J	J	J	J	B	J
border-bottom-width	<Maße>			B	J	B	J*	J	J	B	J	B	J
border-left-width				J	J	B	J*	J	J	J	J	B	J
border-color	<border-top-color>	N	A	B	J	J	J	J	J	B	J	J	J
(Kurzschrift für die Rahmenfarbe)	<border-right-color>			B	J	J	J	J	J	B	J	J	J
	<border-bottom-color>			B	J	J	J	J	J	B	J	J	J
	<border-left-color>			B	J	J	J	J	J	B	J	J	J
border-style	<border-top-style>	N	A	J	J	J	J	J	J	J	J	J	J
(Kurzschrift für den Rahmenstil)	<border-right-style>			J	J	J	J	J	J	J	J	J	J
	<border-bottom-style>			J	J	J	J	J	J	J	J	J	J
	<border-left-style>			J	J	J	J	J	J	J	J	J	J
border-width	<border-top-width>	N	A	J	J	B	J*	J	J	J	J	B	J
(Kurzschrift für die Rahmendicke)	<border-right-width>			J	J	B	J*	J	J	J	J	B	J
	<border-bottom-width>			B	J	B	J*	J	J	B	J	B	J
	<border-left-width>			J	J	B	J*	J	J	J	J	B	J
border	<border-color>	N	A	J	J	J	J	J	J	J	J	J	J
(Kurzschrift für alle Seiten)	<border-style>			J	J	J	J	J	J	J	J	J	J
	<border-width>			J	J	J	J	J	J	J	J	J	J
border-top	border-top-width	N	A	N	J	N	J	J	J	N	J	N	J
(Kurzschrift für die obere Linie)	border-top-style			N	J	N	J	J	J	N	J	N	J
	border-top-color			N	J	N	J	J	J	N	J	N	J
border-right	border-right-width	N	A	N	J	N	J	J	J	N	J	N	J
(Kurzschrift für die rechte Linie)	border-right-style			N	J	N	J	J	J	N	J	N	J
	border-right-color			N	J	N	J	J	J	N	J	N	J
border-bottom	border-bottom-width	N	A	N	J	N	J	J	J	N	J	N	J
(Kurzschrift für die untere Linie)	border-bottom-style			N	J	N	J	J	J	N	J	N	J
	border-bottom-color			N	J	N	J	J	J	N	J	N	J
border-left	border-bottom-width	N	A	N	J	N	J	J	J	N	J	N	J
(Kurzschrift für die linke Linie)	border-bottom-style			N	J	N	J	J	J	N	J	N	J
	border-bottom-color			N	J	N	J	J	J	N	J	N	J

* Nur IE5.5, nicht in IE5.0

Die Eigenschaften zu Größen, Rändern und Abständen finden Sie in den nachfolgenden Tabellen.

Kapitel 7: Elementgrößen und Textumfluss				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
clear	none	N	A	J	J	J	J	J	J	J	J	J	J
	both			J	J	J	J	J	J	J	J	J	J
	left			B	J	N	J*	J	J	B	J	J	J
	right			B	J	N	J*	J	J	B	J	J	J J
float	none	N	A	J	J	J	J*	J	J	J	J	J	J
	left			B	J	B	J*	J	J	B	J	B	J
	right			B	J	B	J	J	J	B	J	B	J
height	auto	N	B	N	J	B	J	J	J	N	J	J	J
	<Maße>			N	J	B	J	J	J	N	J	J	J
width	auto	N	B	B	J	B	J	J	J	B	J	J	J
	<Maße>			B	J	B	J	J	J	B	J	J	J
	<Prozentangaben>			B	J	B	J	J	J	B	J	J	J

* Nur IE5.5, nicht in IE5.0

Kapitel 7: Ränder und Abstände				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
margin-top	auto	N	A	B	J	B	J	J	J	B	J	B	J
margin-right	<Maße>			B	J	B	J	J	J	B	J	B	J
margin-bottom	<Prozentangaben>			B	J	B	J	J	J	B	J	B	J
margin-left				B	J	B	J	J	J	B	J	B	J
margin	siehe margin-top…	N	A	B	J	B	J	J	J	B	J	B	J
padding-top	<Maße>	N	B	B	J	B	J	J	J	B	J	B	J
padding-right	<Prozentangaben>			B	J	B	J	J	J	B	J	B	J
padding-bottom				B	J	B	J	J	J	B	J	B	J
padding-left				B	J	B	J	J	J	B	J	B	J
padding	siehe padding-top…	N	B	B	J	B	J	J	J	B	J	B	J

Kapitel 8: Elemente positionieren				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
position	static	N	A	J	J	J	J	J	J	J	J	J	J
	absolute			J	J	J	J	J	J	J	J	J	J
	relative			J	J	J	J	J	J	J	J	J	J
	fixed			N	N	N	N	N	N	N	N	N	B
left	auto	N	A	J	J	J	J	J	J	J	J	J	J
	<Maße>			J	J	J	J	J	J	J	J	J	J
	<Prozentangaben>			J	J	J	J	J	J	J	J	J	J
top	auto	N	A	J	J	J	J	J	J	J	J	J	J
	<Maße>			J	J	J	J	J	J	J	J	J	J
	<Prozentangaben>			J	J	J	J	J	J	J	J	J	J
bottom	auto	N	A	N	J	N	J	J	J	N	J	N	J
	<Maße>			N	J	N	J	J	J	N	J	N	J
	<Prozentangaben>			N	J	N	J	J	J	N	J	N	J
right	auto	N	A	N	J	N	J	J	J	N	J	N	J
	<Maße>			N	J	N	J	J	J	N	J	N	J
	<Prozentangaben>			N	J	N	J	J	J	N	J	N	J
z-index	auto	N	A	J	J	J	J	J	J	J	J	J	J
	<Ganzzahl>			J	J	J	J	J	J	J	J	J	J

Kapitel 9: Anzeige von Elementen				Windows						Macintosh			
EIGENSCHAFT	WERT	V	T	N4	N6	IE4	IE5	O5	O6	N4	N6	IE4	IE5
clip	auto	N	A	B	J	N	J	B	B	B	J	N	J
	<Pixelangaben für ein Rechteck>			B	J	N	J	B	B	B	J	N	J
display	block	N*	A	B	J	J	J	J	J	B	J	J	J
	inline			N	J	J	J	J	J	N	J	B	J
	list-item			B	J	N	N	J	J	B	J	B	B
	none			J	J	J	J	J	J	J	J	J	J
overflow	auto	N	A	J	J	N	J	J	J	J	J	N	J
	hidden			B	J	N	J	J	J	B	J	N	J
	scroll			N	J	N	J	N	N	N	J	N	J
	visible			J	J	N	J	J	J	J	J	N	J
visibility	hidden	J**	A	J	J	J	J	J	J	J	J	J	J
	visible			J	J	J	J	J	J	J	J	J	J
	inherit			J	J	J	J	J	J	J	J	J	J
	hide			J	N	N	N	N	N	N	J	N	N
	show			J	N	N	N	N	N	N	J	N	N

*Wenn `display` auf `none` gesetzt ist, sind die Kindelemente ebenfalls nicht sichtbar, weil das ganze Elternelement ausgeblendet wird. Ist das Elternelement sichtbar, das Kindelement aber auf `none` gesetzt, dann ist das Elternelement zu sehen, das Kindelement nicht. Die Eigenschaft wird also nicht vererbt.

**Wenn `visibility` auf `inherit` gesetzt ist, nehmen die Kindelemente die Sichtbarkeit des Elternelements an, sonst wird die Eigenschaft nicht vererbt.

INDEX

Symbole

10
 36
<no script> 267
 35
<style> 18

A

Abfrage
  Bildschirminformationen 210
  Browsername 207
  Browserversion 207
  des DOM-Typs 184
  Eigenschaften von Objekten 193
  Farbtiefe 211
  Größe der Anzeigefläche 213
  Größe des geöffneten Browserfenster 212
  Wirkungsweise einer Browserabfrage 185
Absätze
  first-letter 91
  first-line 90
  Pseudo-Formate 90
Abschnitte hervorheben 89
Abstände 113
Adresse eines Objekts 166
Anforderungen an die Navigation 266
Animation 246
Anzeigefläche 130, 213
  Größe mit einer Funktion ändern 250
appCodeName, Objekt navigator 206
appName, Objekt navigator 206
appVersion, Objekt navigator 206
Aufzählungszeichen 79
  eigene Bullets 82
Auswahllisten als Navigation 269
availHeight, Objekt screen 206
availWidth, Objekt screen 206

B

Bausteine für die Navigation 265
beschreibende Angaben 10
beschreibende Schriftwerte 54
Bezugspunkt 135
Bildschirm abfragen 210
Bildschirmauflösung 210
blinkender Text 75
Blocklevel
   36
blocklevel 12
Blocksatz erzwingen 67
bold 57
border 97, 117
Box 97
Breite
  eines Elements 111
  von Objekten, DHTML 195
Browser 4
  abfragen 183
  DHTML-Unterstützung 163
  DOM identifizieren 182
  Informationen auslesen 206
  Internet Explorer 4
  Name abfragen 207
  Netscape 4
  Netscape-Bug bei der DOM-Abfrage 186
  ohne JavaScript-Unterstützung 267
  Opera 4
  Version abfragen 207
  Weiche 182
  Wirkungsweise einer Abfrage 185
Browserabfrage 207
Browserfenster 130
  Bestandteile 130
  Fenster schließen 236
  Fenster verschieben 234
  Größe der Anzeigefläche ändern 250
  neues Fenster öffnen 232
Bullets 78

C

captureEvents 255
Cascading Style Sheets 1
clip.height 195
clip.width 195
Clipping 155
close 236
cm (Zentimeter) 11
colorDepth, Objekt screen 206
Container 14
content 97
cookieEnabled, Objekt navigator 206
CrossBrowser DHTML 162
CrossBrowser DOM 182
  Einsatz 188
  Einsatz der Funktion kind 192
  erweitern für eingebettete Ebenen 190
  Funktionsweise 187
  für Objekte in Frames 238
  ObjID 187
  Parameter 187
  style 187
  um Effektfunktionen erweitern 218
CSS 1
  Browserunterstützung 2
  Dateien importieren 23
  externe Datei 9, 22
  Formate 7
  für verschiedene Medien 42
  Grundlagen 5
  in HTML einbinden 15
  Kommentare 41
  Kurzschrift für Hintergründe 108
  Kurzschrift für Listen 83
  Level 6
  Methoden 9
  Referenz 14
  Syntax 20
  Typen 7
  vererbte Eigenschaften 40
  Versionen 6
CSS und DHTML online 291
CSS-Referenz 2
cursor 84

D

Dateiformat, css 9
Definition 8
DHTML 4
  browserübergreifend 162, 181
  Definition 160
  einfache Techniken 217
  fortgeschrittene Techniken 237
  Grundlagen 159
  Stärken und Schwächen 163
document 215
  body.clientHeight 213
  body.clientWidth 213
  captureEvents 255
  event 255
  in Frames 238
Document Object Model 161
Dokumentation von Funktionen 194
DOM 162, 166
  abfragen 184
  Adresse 166, 187
  All DOM (Internet Explorer 4/5) 178
  CrossBrowser DOM 180, 182
  Einsatz des CrossBrowser DOM 188
  Funktionsweise 175, 187
  gemeinsame Basis 179
  getElementById 179
  ID DOM des W3C 179
  Layer DOM (Netscape 4) 177
  Möglichkeiten 167
  Netscape-Bug bei der Abfrage 186
  Probleme 180
  Typ abfragen 182
  Unterschiede der DOMs 176
dom.js 187
Doppelpunkt 8
Drag&Drop von Objekten 253
Dropdown-Menüs 284
Drucken, CSS optimieren 43
durchgestrichener Text 75

E

Ebenen
- in den Vordergrund holen 225
- Reihenfolge 140
- Reihenfolge dynamisch ändern 225
- überlappen 139
- z-index 140

Editor 3

Effekte
- Animation von Objekten 246
- Anzeige im Frame erzwingen 244
- Anzeige in Frames verhindern 243
- Auswahllisten als Navigation 269
- Drag&Drop von Objekten 253
- Dropdown-Menü 284
- eigene Vor-/Zurück-Links 268
- ein Fenster schließen 236
- ein Fenster verschieben 234
- ein neues Fenster öffnen 232
- Fernsteuerung als Navigation 285
- Grafiken vorausladen 228
- grafische Rolloverleisten 275
- Größe der Anzeigefläche ändern 250
- Mehrere Frames gleichzeitig ändern 245
- Navigation mit dynamischen Frames 276
- Objekte ein- und ausblenden 219
- Objekte in den Vordergrund holen 225
- Objekte verschieben 222
- Rollover 230
- Schatten erzeugen 143

Eigene Blocklevel-Tags 36

Eigene Inline-Tags 35

Eigenschaften
- a:active 87
- a:hover 87
- a:link 87
- a:visited 87
- background 108
- background-attachment 102, 105
- background-color 101
- background-image 102
- background-position 102, 106
- background-repeat 102, 104
- border-bottom 117
- border-color 117
- border-left 117
- border-right 117
- border-style 117
- border-top 117
- border-width 117
- bottom 137
- clear 128
- clip 155
- color 64
- cursor 84
- display 152
- first-letter 89
- first-line 89
- float 127
- height 112
- Kurzschrift für Listen 83
- layer-background-color 123
- left 135
- letter-spacing 70
- line-height 60
- list-style 83
- list-style-image 82
- list-style-position 80
- list-style-type 79
- margin 124
- margin-bottom 124
- margin-left 124
- margin-right 124
- margin-top 124
- overflow 157
- padding 113
- padding-bottom 113
- padding-left 113
- padding-right 113
- padding-top 113
- position 133
- right 136
- Tag-spezifische 38
- text-align 66
- text-decoration 75
- text-indent 72
- text-transform 76
- vererbte 39
- vertical-align 68
- visibility 153
- white-space 73
- width 111
- word-spacing 71
- z-index 140

Ein- und Ausblenden von Objekten 219

eingebetteter Stil 9

Einsatz von Schriften 48

Einzüge 72

Element
Abstände 113
Anzeigeart 151
Bezugspunkt 134
border 97, 118
bottom 137
Box 97
Breite 111
clear 128
clip 156
content 97
display 150
float 127
Größe 110
height 97, 112
Hintergrund 98
Höhe 112
left 135
margin 97, 125
padding 97, 114
position 134
Position oben links 134
Position unten rechts 136
Rahmen 117
Ränder 124
right 136
sichtbarer Bereich 155
Textumfluss 127
Textumfluss beenden 128
top 134
Überhänge 157
überlappen 139
visibility 150
width 97, 111
window 130
Elternelement 134
window 130
em 11
Ereignis 169
überwachen 255
event, Objekt 255
Event-Handler 169
Einsatz mehrerer Handler 173
Einschränkungen bei Netscape 4 170
einsetzen 172
Funktionen aufrufen 172
mehrere Werte in einem Handler 174
Netscape austricksen 171
ex 11
extern 9, 20

F

Farbangaben 10
Farbtiefe 211
Fenster 232
Fenstergröße 210
Fenstergrundlagen 130
Fernsteuerung als Navigation 285
Fette Schrift 57
fixed 133
Flash 163
Flattersatz 67
font-family 52
font-size 53
font-variant, Eigenschaft 58
Format
definieren 7
eigene Blocklevel-Tags 36
eigene Inline-Tags 35
für Pseudo-Elemente 32
Gruppen definieren 34
Klassen für Pseudo-Elemente 92
lokal 17
verschachtelte 33
framedom.js 239
andere Funktionen anpassen 240
Besonderheiten bei Funktionen 242
Frames
allgemeine Informationen 238
angepasstes CrossBrowser DOM 238
Anzeige im Frame erzwingen 244
document 238
ein- und ausblenden 276
mehrere Frames gleichzeitig ändern 245
Seite aus fremden Frames befreien 243
Tipps 243
Funktion
allgemeine Informationen 194
Animation von Objekten 246
Anzeige im Frame erzwingen 244
Anzeige in Frames verhindern 243
aufrufen 172
Auswahllisten als Navigation 269
Bildschirmabfragen 210
Browsername und -version abfragen 207
crossDOM 187
dokumentieren 194
dom.js 187
Drag&Drop von Objekten 253
eigene Vor-/Zurück-Links 268

ein Fenster schließen 236
ein Fenster verschieben 234
ein neues Fenster öffnen 232
Farbtiefe abfragen 211
Fernsteuerung als Navigation 285
frameDOM 239
framedom.js 239
Grafiken vorausladen 228
Größe der Anzeigefläche 213
Größe der Anzeigefläche ändern 250
Größe des geöffneten Browserfensters 212
Höhe und Breite von Objekten 195
kind 190
mehrere Frames gleichzeitig ändern 245
Name einer Funktion 194
Navigation mit dynamischen Frames 276, 285
Objekte ein- und ausblenden 219
Objekte in den Vordergrund holen 225
Objekte verschieben 222
Position eines Objekts 197
Resize-Bug abfangen 214
Rollover-Effekt 230
Sichtbarkeit eines Objekts 201
z-index mit DHTML setzen 204
Funktionsbibliothek 194
Funktionsübersicht anlegen 218
Funktionsweise des CrossBrowser DOM 187

G

getElementById 179
Gif 51
Grafiken
als Hintergrund 102
als Menü 262
in Tabellen 262
Rollover-Effekt 230
Rolloverleisten 275
splitten 262
vorausladen 228
grafische Rolloverleisten 275
Groß-/Kleinschreibung 76
Größe
der Anzeigefläche 213
des geöffneten Browserfensters 212
eines Elements 110
von Objekten, DHTML 195
Grundlagen zu Fenstern 130
Gruppen 34

H

height 97, 112
Objekt screen 206
hidden (Netscape 4) 202
hide (Netscape 4) 202
Hintergründe 98
background-color 101
bei Netscape 99
Farben 101
Grafiken 102
Grafiken kacheln 104
Grafikposition 106
in Inline-Tags 100
Kurzschrift 108
scrollen 105
history 268
Höhe
eines Elements 112
von Objekten, DHTML 195
Hover 230
HTML 1, 2
CSS einbinden 15
externe CSS-Dateien einbinden 22
HTML-Editor 3
HTML-Selektor 7
HTML-Tag 7

I

ID 7
definieren 28
if else-Abfrage 184
in (Inch) 11
Inch 11
Initialen 51
inline 9, 13
intern 9, 18
Internet 1
Internet Explorer 4
Internetseite zum Buch 291
italic 55

J

JavaScript 2
Jpg 51

K

Kapitälchen 58
Klasse 7
  definieren 26
  für Pseudo-Elemente 92
  Unterklassen 31
  verwenden 27
Kommentare 41
  in Funktionen und Scripts 194
Konflikt 7, 24
Kursive Schrift 55
Kurzschrift
  für Abstände (padding) 113
  für Hintergründe 108
  für Rahmen (border) 117
  für Ränder (margin) 124
  für Schriften 61

L

language, Objekt navigator 206
Leerzeichen 73
Links
  a:active 88
  a:hover 88
  a:link 88
  a:visited 88
  eigene Vor-/Zurück-Links 268
  history 268
  Navigationsmenüs 265
  Pseudo-Formate 87
Listen 78
  Aufzählungszeichen 79
  eigene Bulletgrafiken 82
  Kurzschrift 83
  list-style 83
  list-style-image 82
  list-style-position 81
  list-style-type 79
  Position 80
Listentypen
  geordnete Listen 78
  ungeordnete Listen 78
Listing 2
location 215
lokal 9, 16

M

margin 97, 124
Maße 11
Maus-Cursor 84
Mehrere externe CSS-Dateien 24
Mehrere Frames gleichzeitig ändern 245
Menüs 284
Methode
  eingebettet 9
  extern 9
  externe 20
  inline 9
  intern 9, 18
  kombinieren 25
  lokal 9, 16
Millimeter 11
mm (Millimeter) 11
Monospace 49
moveBy 222, 234
moveTo 234

N

Navigation
  allgemeine Hinweise 264
  Anforderungen 266
  Bausteine anlegen 265
  Dropdown-Menü 284
  Fernsteuerung 285
  grafische Rolloverleisten 275
  Mehrere Frames gleichzeitig ändern 245
  mit Auswahllisten 269
  mit dynamischen Frames 276
  Planung 264
  Rollover-Effekte 230
  Rubriken (Hauptbereiche) 265
  Seitenstruktur 264
  Server Side Includes 265
  Struktur 265
navigator 206
Netscape 4
numerische Angaben 10
numerische Werte
  absolut 11
  Maße 11
  ohne Einheiten 11
  Prozentangaben 11
  relativ 11
Nummerierung bei Listen 78

O

Objekt
- animieren 246
- Definition 167
- Drag&Drop 253
- Eigenschaften abfragen 193
- ein- und ausblenden 219
- erzeugen 168
- event 255
- history 268
- Höhe und Breite 195
- ID 167
- in den Vordergrund holen 225
- Name 167
- navigator 206
- Position 197
- Reihenfolge dynamisch ändern 225
- screen 206
- Sichtbarkeit 201, 219
- verschieben 222
- window 206
- z-index 204

Objekt-Hierarchie 166
oblique 55
Öffnen eines neuen Fensters 232
offsetHeight 195
offsetLeft 197
offsetTop 197
offsetWidth 195
onblur 169
onclick 169
onfocus 169
onkeypress 169
onload 169
onmousedown 169
onmouseout 169
onmouseover 169
onmouseup 169
onunload 169
open 232
Opera 4
optimieren 43

P

padding 97, 113
Parameter 187
pc (Pica) 11
Pica 11
Pixel 11
pixelDepth, Objekt screen 206
Planung der Navigation 264
platform, Objekt navigator 206
PopUp 232
Position 134
- oben links 134
- von Objekten, DHTML 197

Positionierung
- absolute 131
- festlegen 134
- fixed 132
- Position 134
- relative 132
- static 131

Proportional-Schriften 49
Prozent 11
Prozentangaben bei Schriften 54
Pseudo-Elemente 32, 86
- Klassen definieren 92

Pseudo-Formate 32, 86
- a:active 87
- a:hover 87
- a:link 87
- a:visited 87
- first-letter 91
- first-line 90

pt (Punkt) 11
Punkt 11
px (Pixel) 11

R

Rahmen 117
- bei Netscape 4 123

Ränder 125
Rautezeichen 10
Rechnerumgebung 193, 206
Reihenfolge 24, 140
- von Objekten 225

relative 133
replaced 13
Resize-Bug im Netscape 214
resizeBy 250
resizeTo 250
return
- false 171, 184
- true 184

Rollover, frameübergreifend 242
Rollover-Effekte 230
Rückgabewerte 184

S

Sans-serif 49
Schatteneffekt 143
Schließen eines Fensters 236
Schmuckschrift 51
Schreibweise 8
Schrift 47
  color 64
  font 61
  font-family 52
  font-size 53
  font-style 55
  font-variant 58
  font-weight 57
  Kurzschrift 61
  line-height 60
  Zeilenhöhen 59
Schriftarten 50
Schriftfamilie 50
  festlegen 52
Schriftfarben 64
Schriftgewicht 57
Schriftgrößen
  erlaubte Werte 54
  festlegen 53
Schrifttypen 49
Schriftzug 51
screen 206
Seitenstruktur 264
Selektor 7
  Formateigenschaften 7
Semikolon 8
Serif 49
Serifenlose Schriften 49
Serifenschrift 49
Server Side Includes 265
show (Netscape 4) 202
Sichtbarer Bereich 155
Sichtbarkeit eines Objekts 201, 219
Software 3
SSI 265
Startpositionen 134
Startpunkt 130
static 131
Stil 8
  Definition 8
  Eigenschaft 8
  Namen 8
  Regel 8
Strichpunkt 8
style (im DOM) 187
style.left 197
style.pixelLeft 197
style.pixelTop 197
style.top 197
Supportseite 2
  zum Buch 291
Syntax
  externe Methode 20
  interne Methode 18
  lokale Methode 16

T

Tag 7, 9, 14
  35, 37
  Blocklevel-Tags 12
  Inline-Tags 12
  Replaced-Tags 12
Text
  Absätze hervorheben 89
  blinkend 75
  durchgestrichen 75
  Einzüge 72
  Groß-/Kleinschreibung 76
  horizontal ausrichten 66
  Leerzeichen 73
  letter-spacing 70
  text-align 66
  text-decoration 75
  text-indent 72
  text-transform 76
  Textumfluss beenden 128
  überstrichen 75
  um Elemente fließen lassen 127
  Umbrüche 73
  unterstrichen 75
  vertical-align 69
  vertikal ausrichten 68
  white-space 73
  word-spacing 71
  Wortabstand 71
  Zeichenabstand 70
Textdekorationen 75
Texteditor 3
Textfarben 64
Textformatierung 65
Textumfluss 127
  beenden 128

U

Überhänge 157
Überlappen 139
Überschriften 51
überstrichener Text 75
Überwachung von Ereignissen 255
unabhängige Formate 7
Unterklassen 30
Unterstreichung 75
unterstrichener Text 75
URL-Angaben 10
Ursprung 130
  eines Frames 130
userAgent, Objekt navigator 206

V

Variable 184
Verschieben
  eines Fensters 234
  von Objekten 222
visibility, mit DHTML festlegen 201
visible (Netscape4) 202
Vorausladen von Grafiken 228
Vorrang 7

W

W3C 1
Webseite
  aus Frames befreien 243
  immer im Frame anzeigen 244
Website zum Buch 2, 291
Weiche 182
Wert 8
  Angaben 10
  beschreibende Angaben 10
  Farbangaben 10
  numerische Angaben 11
  URL-Angaben 10
width 97, 111
  Objekt screen 206
window 130, 166, 206
  close 236
  event 255
  frames 238
  history 268
  innerHeight 213
  innerWidth 213
  moveBy 234
  moveTo 234
  open 232
  outerHeight 212
  outerWidth 212
  resizeBy 250
  resizeTo 250
Wirkungsweise einer Browserabfrage 185
World Wide Web 1
Wortabstand 71
WWW 1

Z

Zahlen 11
Zeichenabstand 70
Zeilenhöhen 59
Zeilenumbruch 17
  festlegen 73
Zentimeter 11
z-index 140
  dynamisch ändern 225
  mit DHTML setzen 204
zIndex (DHTML) 204
Zugangsdaten für die Internetseite 291