



# Vorwort

Das Gebiet der Web-Technologien – also der Sprachen, Übertragungsprotokolle und Schnittstellen, die dafür sorgen, dass Web-Seiten ins Browser-Fenster des Anwenders gelangen – ist mittlerweile unüberschaubar groß. Einige davon bauen aufeinander auf, andere konkurrieren miteinander. Neueinsteiger stellen deshalb häufig Fragen wie die, womit sie eigentlich anfangen sollen, welche Sprachen zu lernen sich am ehesten lohnt, in welcher Reihenfolge man vorgehen sollte usw.

Fast ebenso unüberschaubar wie das Gebiet der Web-Technologien ist mittlerweile aber auch das Angebot an Einzelpublikationen zu bestimmten Technologien. Angesichts dieser Situation ist es wichtig, den Überblick zu behalten. Zusammenhänge zwischen den einzelnen Web-Technologien müssen verstanden werden, grundlegende Unterschiede wie der zwischen serverseitiger und clientseitiger Datenverarbeitung müssen begriffen werden.

Das vorliegende Buch, das in der vollständigen Ausgabe aus zwei Bänden besteht, ist eine Gesamtschau auf die heutige Situation beim Web Publishing. Der erste Band behandelt die Basis-Technologien HTML, CSS Stylesheets und JavaScript/DHTML sowie einführende Abschnitte für Einsteiger. Der zweite Band behandelt weiterführende Technologien wie das Definieren eigener Beschreibungssprachen mit XML und serverseitige Datenverarbeitung mit CGI/Perl. Trotz seines Umfangs kann auch die gesamte zweibändige Ausgabe nicht alles behandeln, was heute verbreitet ist. So werden beispielsweise PHP, die Microsoft-orientierte ASP-Technologie, die neueren JavaServerPages (JSP) oder proprietäre, aber beliebte Themen wie Macromedia Flash ausgeklammert. Dennoch ist das Buch wohl eines der umfassendsten Werke im Bereich der Web-Technologien auf dem Markt. Es richtet sich sowohl an Einsteiger, die Schritt für Schritt in einer empfohlenen Reihenfolge an die beschriebenen Technologien herangeführt werden, als auch an Profis, die eher etwas zum schnellen Nachschlagen benötigen.

Eine Besonderheit dieses Buches ist, dass sein zugrunde liegender Text im Web bereits berühmt und tausendfach verlinkt ist. Das Buch basiert nämlich auf der im Web frei erhältlichen Dokumentation SELFHTML. Diese Dokumentation hat im deutschsprachigen Raum eine siebenstelligen Leserzahl, gilt als anerkanntes Lehrwerk an Schulen, Berufsschulen und Universitäten, und ist in vielen Intranets, Web-Agenturen und Software-Schmieden längst zu einer unentbehrlichen Basishilfe geworden. SELFHTML (auf der CD zu diesem Buch enthalten) ist ein Hypertext-Werk, das selbst in HTML geschrieben ist und in mehreren Jahren durch Feedback und anhaltendes Autoreninteresse gereift ist. Viel Anwender-Feedback und die Unterstützung durch engagierte, dem Autor nahe stehende Fachleute sind in den Text mit eingeflossen.

Von dieser Praxisnähe profitiert natürlich auch die Print-Ausgabe. Der Buchinhalt ist keine Auftragsarbeit des herausgebenden Franzis Verlages, sondern eine zeitlich und inhaltlich ungebundene Eigenleistung des Autors und seiner Unterstützer.

Der Autor bedankt sich besonders bei folgenden Personen, die zur inhaltlichen Qualitätssicherung von SELFHTML und damit auch des Textes der beiden Bände dieses Buches mit beigetragen haben:

Antje Hofmann (<http://www.pc-anfaenger.de/>), Thomas J. Sebestyen (<http://www.meta-text.net/>), Burkhard Becker (<http://www.bubec.de/>), Swen Wacker (<http://www.mixtura.de/>) und André Malo (<http://www.o3media.de/>).

Der Dank geht ferner an viele andere, die durch Fehlermeldungen, andere Zusarbeiten oder fachlich besonders produktives Engagement im Aktuell-Raum des SELFHTML-Projekts (<http://selfaktuell.teamone.de/>) dazu beigetragen haben und weiter dazu beitragen, dass das Projekt weiterlebt und damit auch die Basis zur Weiterentwicklung der vorliegenden Print-Ausgabe bilden kann. Genannt werden diese Personen auf den Seiten <http://selfaktuell.teamone.de/people/devs.htm> und <http://selfaktuell.teamone.de/people/devs.htm>.

Dank geht ferner an Wolfgang Nefzger, der die Buchumsetzung von SELFHTML besorgte, sowie an den Franzis Verlag, bei dem sich der Autor gut aufgehoben fühlt.

München, im Februar 2002

Stefan Münz

## Verwendete Symbole



Das beschriebene Feature gehört seit der Version 2.0 zum HTML-Standard.



Das beschriebene Feature gehört seit der Version 3.2 zum HTML-Standard.



Das beschriebene Feature gehört seit der Version 4.0 zum HTML-Standard.



Das beschriebene Feature gehört zur Version 1.0 des XHTML-Standard.



Symbol für *deprecated*. Bedeutet, dass das beschriebene Feature noch zum HTML-Standard gehört, in Zukunft jedoch entfallen soll. In der Regel gibt es andere HTML-Elemente oder andere Ergänzungssprachen wie CSS Stylesheets, durch die das entsprechende Feature in Zukunft realisiert werden soll.



Das beschriebene Feature gehört seit der Version 1.0 zum CSS-Standard.



Das beschriebene Feature gehört seit der Version 2.0 zum CSS-Standard.



Das beschriebene Feature gehört seit der Version 1.0 zum XML-Standard.



Das beschriebene Feature gehört seit der Version 1.0 zum XSLT-Standard.



Das beschriebene Feature gehört seit der Version 1.0 zum XPath-Standard.



Das beschriebene Feature gehört seit der Version 1.0 zum DOM-Standard (Document Object Model).



Das beschriebene Feature gehört seit der Version 2.0 zum DOM-Standard (Document Object Model).



Das beschriebene Feature ist seit der Version 1.0 von JavaScript verfügbar.



Das beschriebene Feature ist seit der Version 1.1 von JavaScript verfügbar.



Das beschriebene Feature ist seit der Version 1.2 von JavaScript verfügbar.



Das beschriebene Feature ist seit der Version 1.3 von JavaScript verfügbar.



Das beschriebene Feature ist seit der Version 1.5 von JavaScript verfügbar.



Das beschriebene Feature unterstützt der MS Internet Explorer ab Version 1.0.





Das beschriebene Feature unterstützt der MS Internet Explorer ab Version 2.0.



Das beschriebene Feature unterstützt der MS Internet Explorer ab Version 3.0.



Das beschriebene Feature unterstützt der MS Internet Explorer ab Version 4.0.



Das beschriebene Feature unterstützt der MS Internet Explorer ab Version 5.0.



Das beschriebene Feature unterstützt der MS Internet Explorer ab Version 5.0 oder 5.5.



Das beschriebene Feature unterstützt Netscape seit Version 1.0.



Das beschriebene Feature unterstützt Netscape seit Version 1.1.



Das beschriebene Feature unterstützt Netscape seit Version 2.0.



Das beschriebene Feature unterstützt Netscape seit Version 3.0.



Das beschriebene Feature unterstützt Netscape seit Version 4.0.



Das beschriebene Feature unterstützt Netscape seit Version 4.01, 4.5, 4.6, 4.7.



Das beschriebene Feature unterstützt Netscape seit Version 6.0.

# Inhaltsverzeichnis

|          |                                              |           |
|----------|----------------------------------------------|-----------|
| <b>1</b> | <b>Internet und WWW .....</b>                | <b>27</b> |
| 1.1      | Entstehung des Internet .....                | 27        |
| 1.1.1    | Das ARPA-Net .....                           | 27        |
| 1.1.2    | Wissenschaftliche Einrichtungen .....        | 28        |
| 1.1.3    | Das Netz der Netze .....                     | 28        |
| 1.2      | Basis-Standards im Internet .....            | 29        |
| 1.2.1    | Das TCP/IP-Protokoll .....                   | 29        |
| 1.2.2    | IP-Adressierung .....                        | 30        |
| 1.2.3    | Client-Server-Technologie .....              | 31        |
| 1.2.4    | DNS – Domain Name Service .....              | 32        |
| 1.2.5    | Routing und Gateways .....                   | 34        |
| 1.2.6    | Selbstorganisation im Internet .....         | 34        |
| 1.3      | Dienste im Internet .....                    | 36        |
| 1.3.1    | E-Mail .....                                 | 36        |
| 1.3.2    | Telnet und SSH .....                         | 37        |
| 1.3.3    | File Transfer (FTP) .....                    | 38        |
| 1.3.4    | Gopher .....                                 | 39        |
| 1.3.5    | Chat (IRC) .....                             | 39        |
| 1.3.6    | Newsgroups (News) .....                      | 40        |
| 1.3.7    | World Wide Web (WWW) .....                   | 41        |
| 1.4      | Entstehung des World Wide Web .....          | 41        |
| 1.4.1    | Tim Berners-Lee und die Anfänge .....        | 41        |
| 1.4.2    | Marc Andreessen, Netscape und der Boom ..... | 43        |
| 1.4.3    | Die Arbeit des W3-Konsortiums .....          | 44        |
| 1.5      | Hypertext .....                              | 45        |
| 1.5.1    | Definitionen zu »Hypertext« .....            | 45        |
| 1.5.2    | Begriffe rund um Hypertext .....             | 49        |
| 1.5.3    | Geschichte von Hypertext .....               | 54        |
| 1.5.4    | Hypertext und menschliches Verstehen .....   | 57        |
| 1.6      | Hilfsmittel .....                            | 60        |
| 1.6.1    | Dokumente und Dokumentationen .....          | 60        |
| 1.6.2    | Software .....                               | 65        |
| 1.6.3    | Services im Web .....                        | 68        |
| 1.6.4    | Support .....                                | 69        |

|          |                                                            |           |
|----------|------------------------------------------------------------|-----------|
| <b>2</b> | <b>Web-Technologien .....</b>                              | <b>73</b> |
| 2.1      | HTML .....                                                 | 73        |
| 2.1.1    | HTML als »lingua franca« des Web .....                     | 73        |
| 2.1.2    | HTML als softwareunabhängiges Klartextformat .....         | 74        |
| 2.1.3    | HTML als Auszeichnungssprache .....                        | 74        |
| 2.1.4    | HTML für Hypertext .....                                   | 75        |
| 2.1.5    | HTML-Versionen .....                                       | 75        |
| 2.2      | CSS Stylesheets .....                                      | 77        |
| 2.2.1    | Formatsprache für HTML-Elemente .....                      | 77        |
| 2.2.2    | CSS-Versionen.....                                         | 78        |
| 2.3      | XML und XML-Derivate.....                                  | 79        |
| 2.3.1    | XML als Definitionssprache für Auszeichnungssprachen ..... | 79        |
| 2.3.2    | XML-Derivate.....                                          | 80        |
| 2.3.3    | XML-Versionen .....                                        | 81        |
| 2.4      | JavaScript/DOM .....                                       | 81        |
| 2.4.1    | Web-Seiten als Anwendungen .....                           | 81        |
| 2.4.2    | JavaScript-Informationen .....                             | 82        |
| 2.5      | CGI und Perl .....                                         | 83        |
| 2.5.1    | CGI-Schnittstelle und Perl-Interpreter .....               | 83        |
| 2.5.2    | CGI/Perl-Informationen .....                               | 84        |
| 2.6      | PHP.....                                                   | 84        |
| 2.6.1    | PHP und HTML .....                                         | 84        |
| 2.6.2    | PHP-Informationen.....                                     | 85        |
| 2.7      | ASP .....                                                  | 85        |
| 2.7.1    | ASP und HTML.....                                          | 85        |
| 2.7.2    | ASP-Informationen .....                                    | 86        |
| 2.8      | Java.....                                                  | 86        |
| 2.8.1    | Java und HTML .....                                        | 86        |
| 2.8.2    | Java-Informationen.....                                    | 87        |
| 2.9      | ActiveX .....                                              | 88        |
| 2.9.1    | ActiveX und HTML.....                                      | 88        |
| 2.9.2    | ActiveX-Informationen .....                                | 88        |
| 2.10     | Flash .....                                                | 89        |
| 2.10.1   | Flash und HTML.....                                        | 89        |
| 2.10.2   | Flash-Informationen .....                                  | 90        |
| <b>3</b> | <b>Allgemeine Regeln für HTML.....</b>                     | <b>91</b> |
| 3.1      | Textauszeichnung .....                                     | 91        |
| 3.1.1    | Elemente und Tags in HTML .....                            | 91        |
| 3.1.2    | Verschachtelung von Elementen.....                         | 92        |
| 3.1.3    | Attribute in Tags .....                                    | 92        |
| 3.1.4    | HTML-Parser .....                                          | 93        |
| 3.2      | Grundgerüst einer (X)HTML-Datei.....                       | 93        |
| 3.2.1    | Grundgerüst einer HTML-Datei .....                         | 94        |

|          |                                                               |            |
|----------|---------------------------------------------------------------|------------|
| 3.2.2    | Grundgerüst einer XHTML-Datei .....                           | 94         |
| 3.2.3    | Dokumenttyp-Angaben .....                                     | 95         |
| 3.2.4    | Hinweis für die Praxis .....                                  | 98         |
| 3.3      | Regeln beim Editieren von HTML .....                          | 98         |
| 3.4      | Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen .....    | 99         |
| 3.4.1    | Allgemeines zu Zeichen in HTML .....                          | 99         |
| 3.4.2    | Deutsche Umlaute und scharfes S .....                         | 100        |
| 3.4.3    | Das Euro-Zeichen in HTML .....                                | 101        |
| 3.4.4    | HTML-eigene Zeichen maskieren .....                           | 101        |
| 3.5      | Farben definieren in HTML .....                               | 101        |
| 3.5.1    | Hexadezimale Angabe von Farben .....                          | 102        |
| 3.5.2    | Farbnamen für die 16 Grundfarben .....                        | 104        |
| 3.5.3    | Netscape-Farbnamen .....                                      | 104        |
| 3.6      | Kommentare und Credits .....                                  | 107        |
| 3.6.1    | Nicht angezeigter Text (Kommentar) .....                      | 107        |
| 3.6.2    | Credits .....                                                 | 108        |
| 3.7      | Referenzieren in HTML .....                                   | 108        |
| 3.7.1    | Mit vollständigen URIs referenzieren .....                    | 108        |
| 3.7.2    | Absolute Pfadangaben relativ zum Basis-URI .....              | 110        |
| 3.7.3    | Pfadangaben relativ zum Basis-URI .....                       | 110        |
| 3.8      | Konventionen für Dateinamen .....                             | 111        |
| 3.8.1    | Dateinamen im Hinblick auf Server-Rechner .....               | 111        |
| 3.8.2    | Dateinamen im Hinblick auf Kompatibilität .....               | 112        |
| 3.8.3    | Dateiendungen .....                                           | 112        |
| 3.8.4    | Default-Dateinamen bei Web-Servern .....                      | 112        |
| 3.9      | Informationsverteilung und Dateioorganisation .....           | 113        |
| 3.9.1    | Hypertext schreiben – Information verteilen .....             | 113        |
| 3.9.2    | Dateioorganisation .....                                      | 113        |
| 3.10     | Guter HTML-Stil .....                                         | 114        |
| 3.10.1   | Nicht für bestimmten Browser schreiben .....                  | 115        |
| 3.10.2   | Nicht für bestimmte Bildschirmauflösung schreiben .....       | 116        |
| 3.10.3   | HTML-Elemente nicht zweckentfremden .....                     | 116        |
| 3.10.4   | Ehrliche und aussagekräftige Verweistexte verwenden .....     | 117        |
| 3.10.5   | Grafiken richtig einsetzen .....                              | 117        |
| <b>4</b> | <b>Dateiweite Einstellungen .....</b>                         | <b>119</b> |
| 4.1      | HTML-Kopfdaten .....                                          | 119        |
| 4.1.1    | Titel einer HTML-Datei .....                                  | 119        |
| 4.1.2    | Meta-Angaben zum Inhalt .....                                 | 119        |
| 4.1.3    | Angabe zum Default-Zeichensatz .....                          | 125        |
| 4.1.4    | Angaben zu Default-Sprachen für Scripts und Stylesheets ..... | 126        |
| 4.1.5    | Datei von Originaladresse laden .....                         | 126        |
| 4.1.6    | Weiterleitung zu anderer Adresse (Forwarding) .....           | 127        |
| 4.1.7    | Angaben zu PICS .....                                         | 128        |

|          |                                                         |            |
|----------|---------------------------------------------------------|------------|
| 4.1.8    | Meta-Angaben in separater Datei (Profil) .....          | 129        |
| 4.1.9    | Schema zu einer Meta-Angabe .....                       | 129        |
| 4.1.10   | Diverse Meta-Angaben .....                              | 130        |
| 4.1.11   | Adressbasis und Zielfensterbasis .....                  | 130        |
| 4.1.12   | Logische Beziehungen .....                              | 132        |
| 4.1.13   | Durchsuchbarkeit mit Server-Kommunikation .....         | 134        |
| 4.2      | Dateiweite Einstellungen .....                          | 135        |
| 4.2.1    | Farben für Hintergrund, Text und Verweise .....         | 135        |
| 4.2.2    | Hintergrundbild (Wallpaper) .....                       | 137        |
| 4.2.3    | Seitenränder definieren .....                           | 138        |
| 4.2.4    | Hintergrundmusik .....                                  | 140        |
| <b>5</b> | <b>Text strukturieren .....</b>                         | <b>141</b> |
| 5.1      | Überschriften .....                                     | 141        |
| 5.1.1    | Überschriften definieren .....                          | 141        |
| 5.1.2    | Überschriften ausrichten .....                          | 142        |
| 5.1.3    | Überschriften formatieren mit CSS .....                 | 143        |
| 5.2      | Textabsätze .....                                       | 144        |
| 5.2.1    | Textabsätze definieren .....                            | 144        |
| 5.2.2    | Textabsätze ausrichten .....                            | 145        |
| 5.2.3    | Textabsätze formatieren mit CSS .....                   | 146        |
| 5.3.4    | Zeilenumbruch .....                                     | 147        |
| 5.3.5    | Zeilenumbruch erzwingen .....                           | 147        |
| 5.3.6    | Automatischen Zeilenumbruch verhindern .....            | 148        |
| 5.3.7    | Geschützte Leerzeichen .....                            | 149        |
| 5.3.8    | Zeilenumbruch erlauben .....                            | 149        |
| 5.4      | Listen .....                                            | 150        |
| 5.4.1    | Aufzählungsliste definieren .....                       | 150        |
| 5.4.2    | Nummerierte Liste definieren .....                      | 152        |
| 5.4.3    | Definitionsliste definieren .....                       | 153        |
| 5.4.4    | Verzeichnis- oder Menüliste definieren .....            | 154        |
| 5.4.5    | HTML-Eigenschaften für Listen .....                     | 155        |
| 5.4.6    | Listen formatieren mit CSS .....                        | 157        |
| 5.5      | Zitate und Adressen .....                               | 159        |
| 5.5.1    | Zitate definieren .....                                 | 159        |
| 5.5.2    | Zitate mit URI-Quellenangabe .....                      | 160        |
| 5.5.3    | Adressen definieren .....                               | 160        |
| 5.5.4    | Zitate und Adressen formatieren mit CSS .....           | 161        |
| 5.6      | Präformatierter Text (wie im Editor eingegeben) .....   | 163        |
| 5.6.1    | Textabschnitt mit präformatiertem Text definieren ..... | 163        |
| 5.6.2    | Breite eines Bereichs mit präformatiertem Text .....    | 164        |
| 5.6.3    | Ältere HTML-Elemente für präformatierten Text .....     | 165        |
| 5.6.4    | Präformatierten Text formatieren mit CSS .....          | 165        |
| 5.7      | Logische Auszeichnungen im Text .....                   | 167        |
| 5.7.1    | HTML-Elemente für logische Auszeichnung im Text .....   | 167        |

|          |                                                                           |            |
|----------|---------------------------------------------------------------------------|------------|
| 5.7.2    | Änderungsmarkierungen im Dokument.....                                    | 169        |
| 5.7.3    | Textrichtung ändern .....                                                 | 170        |
| 5.7.4    | Logische Elemente für Textauszeichnung mit CSS formatieren .....          | 171        |
| 5.8      | Physische Auszeichnungen im Text.....                                     | 171        |
| 5.8.1    | HTML-Elemente für physische Auszeichnung im Text.....                     | 172        |
| 5.8.2    | Physische Elemente für Textauszeichnung zusätzlich mit CSS formatieren .. | 173        |
| 5.9      | Allgemeine Elemente für Textbereiche .....                                | 174        |
| 5.9.1    | Allgemeines Block-Element.....                                            | 174        |
| 5.9.2    | Älteres Block-Element für zentrierte Bereiche .....                       | 175        |
| 5.9.3    | Allgemeines Inline-Element.....                                           | 176        |
| 5.9.4    | Allgemeine Elemente formatieren mit CSS.....                              | 176        |
| 5.10     | Trennlinien .....                                                         | 178        |
| 5.10.1   | Trennlinien definieren .....                                              | 178        |
| 5.10.2   | Trennlinien mit HTML gestalten .....                                      | 179        |
| 5.10.3   | Trennlinien mit CSS gestalten.....                                        | 180        |
| 5.11     | Ältere Elemente zur Schriftformatierung .....                             | 181        |
| 5.11.1   | Schriftart, Schriftgröße und Schriftfarbe.....                            | 181        |
| 5.11.2   | CSS-Entsprechungen .....                                                  | 183        |
| 5.12     | Ältere Browser-proprietäre Elemente.....                                  | 184        |
| 5.12.1   | Lauftext – Marquee (Microsoft) .....                                      | 184        |
| 5.12.2   | Mehrspaltiger Textfluss (Netscape).....                                   | 186        |
| 5.12.3   | Raumschaffer (Netscape).....                                              | 187        |
| 5.12.4   | Blinkender Text (Netscape) .....                                          | 188        |
| <b>6</b> | <b>Verweise.....</b>                                                      | <b>189</b> |
| 6.1      | Verweise definieren und gestalten .....                                   | 189        |
| 6.1.1    | Allgemeines zu Verweisen .....                                            | 189        |
| 6.1.2    | Beispiele für Verweise .....                                              | 190        |
| 6.1.3    | Verweise optisch gestalten mit CSS .....                                  | 191        |
| 6.1.4    | Zielfenster für Verweise bestimmen .....                                  | 193        |
| 6.2      | Projektinterne Verweise .....                                             | 194        |
| 6.2.1    | Verweise zu Dateien oder Quellen im Projekt.....                          | 194        |
| 6.2.2    | Anker definieren und Verweise zu Ankern .....                             | 196        |
| 6.3      | Projektexterne Verweise .....                                             | 198        |
| 6.3.1    | Allgemeines zu projektexternen Verweisen .....                            | 198        |
| 6.3.2    | Beispiele für projektexterne Verweise .....                               | 198        |
| 6.4      | E-Mail-Verweise.....                                                      | 200        |
| 6.4.1    | Verweis zu E-Mail-Adresse definieren .....                                | 200        |
| 6.4.2    | Optionen bei E-Mail-Verweisen.....                                        | 202        |
| 6.5      | Download-Verweise und Verweise auf beliebige Dateien.....                 | 205        |
| 6.5.1    | Download-Verweise .....                                                   | 205        |
| 6.5.2    | Verweise zu beliebigen Dateien .....                                      | 206        |
| 6.5.3    | Mime-Type des Verweisziels angeben.....                                   | 207        |
| 6.6      | Typisierte Verweise.....                                                  | 208        |

|          |                                                          |            |
|----------|----------------------------------------------------------|------------|
| 6.6.1    | Logische Beziehung zum Verweisziel angeben .....         | 208        |
| 6.6.2    | Sprache und Zeichensatz des Verweisziels angeben .....   | 209        |
| 6.7      | Tabulator-Reihenfolge und Tastaturkürzel .....           | 210        |
| 6.7.1    | Tabulator-Reihenfolge für Verweise .....                 | 210        |
| 6.7.2    | Tastaturkürzel für Verweise .....                        | 211        |
| <b>7</b> | <b>Grafiken .....</b>                                    | <b>213</b> |
| 7.1      | Grafiken einbinden .....                                 | 213        |
| 7.1.1    | Allgemeines zu Grafiken .....                            | 213        |
| 7.1.2    | Grafikreferenz definieren .....                          | 213        |
| 7.1.3    | Breite und Höhe von Grafiken .....                       | 216        |
| 7.1.4    | Rahmen um Grafiken .....                                 | 217        |
| 7.1.5    | Namen für Grafiken .....                                 | 218        |
| 7.1.6    | Langbeschreibung für Grafiken .....                      | 218        |
| 7.2      | Grafiken ausrichten .....                                | 219        |
| 7.2.1    | Grafik zum umgebenden Text ausrichten .....              | 219        |
| 7.2.2    | Text um Grafik fließen lassen .....                      | 221        |
| 7.2.3    | Grafik und andere Inhalte mit Tabellen ausrichten .....  | 223        |
| 7.2.4    | Grafiken ausrichten mit CSS Stylesheets .....            | 224        |
| 7.3      | Grafiken als Verweise .....                              | 227        |
| 7.3.1    | Grafiken anstelle von Verweistext definieren .....       | 227        |
| 7.4      | Verweissensitive Grafiken (Image Maps) .....             | 228        |
| 7.4.1    | Verweissensitive Grafiken definieren .....               | 228        |
| 7.4.2    | Server-seitige verweissensitive Grafiken .....           | 232        |
| <b>8</b> | <b>Tabellen .....</b>                                    | <b>233</b> |
| 8.1      | Aufbau einer Tabelle .....                               | 233        |
| 8.1.1    | Tabellen definieren .....                                | 233        |
| 8.1.2    | Spalten vordefinieren .....                              | 235        |
| 8.1.3    | Kopf, Körper und Fuß einer Tabelle definieren .....      | 238        |
| 8.2      | Gestaltung einer Tabelle .....                           | 240        |
| 8.2.1    | Zellenabstand und Zelleninnenabstand .....               | 240        |
| 8.2.2    | Regeln für den Außenrahmen .....                         | 242        |
| 8.2.3    | Regeln für Gitternetzlinien .....                        | 243        |
| 8.2.4    | Breiten- und Höhenangaben .....                          | 245        |
| 8.2.5    | Zeilenumbruch in Zellen verhindern .....                 | 248        |
| 8.2.6    | Ausrichtung von Zellen .....                             | 249        |
| 8.2.7    | Hintergrundfarben und Hintergrundbilder .....            | 251        |
| 8.2.8    | Farben für Rand und Gitternetzlinien (Microsoft) .....   | 254        |
| 8.3      | Tabellen mit CSS gestalten .....                         | 254        |
| 8.4      | Zellen verbinden .....                                   | 254        |
| 8.4.1    | Zellen in einer Zeile spaltenweise verbinden .....       | 254        |
| 8.4.2    | Zellen in einer Spalte zeilenweise verbinden .....       | 255        |
| 8.4.3    | Zellen zugleich spalten- und zeilenweise verbinden ..... | 256        |

|           |                                                              |            |
|-----------|--------------------------------------------------------------|------------|
| 8.5       | Tabellenbeschriftung und Tabellenausrichtung .....           | 257        |
| 8.5.1     | Tabellenüberschrift/-unterschrift definieren.....            | 257        |
| 8.5.2     | Tabellenausrichtung .....                                    | 259        |
| 8.5.3     | Weitere Informationen .....                                  | 261        |
| 8.6       | Tabellen als Mittel für Web-Seitenlayouts.....               | 261        |
| 8.6.1     | Allgemeines zu Tabellen für Web-Seitenlayouts.....           | 261        |
| 8.6.2     | Beispiel eines typischen Tabellen-Layouts.....               | 261        |
| 8.7       | Tabellen und nichtvisuelle Medien.....                       | 265        |
| 8.7.1     | Allgemeines .....                                            | 265        |
| 8.7.2     | Zusammenfassung des Tabelleninhalts .....                    | 265        |
| 8.7.3     | Kurzbeschreibung für Zellen definieren.....                  | 266        |
| 8.7.4     | Kopfzellen in Zellen referenzieren .....                     | 267        |
| 8.7.5     | Tabellen-Kategorien definieren .....                         | 269        |
| <b>9</b>  | <b>Frames – mehrere Fenster .....</b>                        | <b>271</b> |
| 9.1       | Framesets und Frames definieren.....                         | 271        |
| 9.1.1     | Allgemeines zu Frames .....                                  | 271        |
| 9.1.2     | Grundgerüst einer HTML-Datei mit Framesets .....             | 272        |
| 9.1.3     | Framesets definieren .....                                   | 273        |
| 9.1.4     | Frames zu einem Frameset definieren .....                    | 276        |
| 9.1.5     | Noframes-Bereich definieren .....                            | 279        |
| 9.1.6     | Langbeschreibung für Frame-Fenster .....                     | 281        |
| 9.2       | Eigenschaften von Frames .....                               | 281        |
| 9.2.1     | Bildlaufleisten erzwingen/verhindern .....                   | 281        |
| 9.2.2     | Mindestabstände zwischen Fensterrand und Fensterinhalt ..... | 282        |
| 9.2.3     | Unveränderbare Fenstergröße .....                            | 284        |
| 9.2.4     | Rahmendicke/Unsichtbare Fensterrahmen .....                  | 284        |
| 9.2.5     | Farbige Fensterrahmen.....                                   | 287        |
| 9.3       | Verweise bei Frames .....                                    | 287        |
| 9.3.1     | Verweise zu anderen Frames.....                              | 287        |
| 9.3.2     | Verweise zum Beenden von Framesets.....                      | 288        |
| 9.4       | Eingebettete Frames.....                                     | 289        |
| 9.4.1     | Eingebettete Frames definieren .....                         | 289        |
| 9.4.2     | Eigenschaften eingebetteter Frames .....                     | 291        |
| 9.4.3     | Verweise zu eingebetteten Frames.....                        | 292        |
| 9.5       | Frames als Mittel für Seitenlayouts.....                     | 294        |
| 9.5.1     | Sinnvolle Einsatzmöglichkeiten für Frames .....              | 294        |
| 9.5.2     | Unfaire Schaulaufenster-Effekte .....                        | 296        |
| <b>10</b> | <b>Formulare.....</b>                                        | <b>297</b> |
| 10.1      | Formulare definieren .....                                   | 297        |
| 10.1.1    | Allgemeines zu Formularen .....                              | 297        |
| 10.1.2    | Formularbereich definieren.....                              | 298        |
| 10.1.3    | Zielfenster für Server-Antwort.....                          | 300        |
| 10.1.4    | Zu verarbeitende Zeichensätze .....                          | 300        |



|           |                                                                |            |
|-----------|----------------------------------------------------------------|------------|
| 10.1.5    | Weitere Informationen .....                                    | 301        |
| 10.2      | Eingabefelder und Eingabebereiche.....                         | 301        |
| 10.2.1    | Einzeilige Eingabefelder definieren.....                       | 301        |
| 10.2.2    | Textvorbelegung bei einzeiligen Eingabefeldern .....           | 303        |
| 10.2.3    | Eingabefelder für Passwörter .....                             | 304        |
| 10.2.4    | Mehrzeilige Eingabebereiche definieren.....                    | 305        |
| 10.2.5    | Textvorbelegung bei mehrzeiligen Eingabebereichen .....        | 306        |
| 10.2.6    | Zeilenumbruch bei mehrzeiligen Eingabebereichen .....          | 307        |
| 10.2.7    | Eingabefelder und Eingabebereiche auf »nur lesen« setzen ..... | 309        |
| 10.3      | Auswahllisten .....                                            | 310        |
| 10.3.1    | Auswahllisten definieren .....                                 | 310        |
| 10.3.2    | Auswahllisten mit Mehrfachauswahl .....                        | 312        |
| 10.3.3    | Einträge vorselektieren .....                                  | 313        |
| 10.3.4    | Absendewert von Einträgen bestimmen.....                       | 314        |
| 10.3.5    | Verschachtelte Auswahllisten definieren.....                   | 315        |
| 10.4      | Radiobuttons und Checkboxes .....                              | 317        |
| 10.4.1    | Radiobuttons definieren .....                                  | 317        |
| 10.4.2    | Checkboxes definieren .....                                    | 318        |
| 10.5      | Klick-Buttons.....                                             | 319        |
| 10.5.1    | Klick-Buttons definieren (herkömmlich) .....                   | 319        |
| 10.5.2    | Klick-Buttons definieren (modern) .....                        | 320        |
| 10.6      | Felder für Datei-Upload.....                                   | 322        |
| 10.7      | Versteckte Formularelemente .....                              | 324        |
| 10.7.1    | Versteckte Elemente definieren .....                           | 324        |
| 10.8      | Guppierung von Elementen und Label für Elemente .....          | 325        |
| 10.8.1    | Elemente gruppieren.....                                       | 325        |
| 10.8.2    | Label für Elemente .....                                       | 327        |
| 10.9      | Tabulator-Reihenfolge, Tastaturkürzel und Ausgrauen.....       | 328        |
| 10.9.1    | Tabulator-Reihenfolge .....                                    | 328        |
| 10.9.2    | Tastaturkürzel .....                                           | 329        |
| 10.9.3    | Elemente ausgrauen.....                                        | 330        |
| 10.10     | Buttons zum Absenden oder Abbrechen.....                       | 331        |
| 10.10.1   | Buttons definieren (herkömmlich) .....                         | 331        |
| 10.10.2   | Grafische Buttons definieren.....                              | 333        |
| 10.10.3   | Buttons zum Absenden oder Abbrechen definieren (modern) .....  | 334        |
| 10.11     | Formulare formatieren mit CSS.....                             | 334        |
| 10.11.1   | Formulare mit CSS gestalten .....                              | 334        |
| 10.12     | Formulare verarbeiten .....                                    | 338        |
| 10.12.1   | Formulare als E-Mail .....                                     | 338        |
| 10.12.2   | Formulare und CGI .....                                        | 339        |
| <b>11</b> | <b>Multimedia .....</b>                                        | <b>341</b> |
| 11.1      | Objekte einbinden .....                                        | 341        |
| 11.1.1    | Allgemeines zu Objekten in HTML.....                           | 341        |

|           |                                                                  |            |
|-----------|------------------------------------------------------------------|------------|
| 11.1.2    | Datendateien als Objekt einbinden .....                          | 342        |
| 11.1.3    | Verweissensitive Grafiken als Objekt einbinden .....             | 344        |
| 11.1.4    | Java-Applets als Objekt einbinden .....                          | 345        |
| 11.1.5    | ActiveX-Controls als Objekt einbinden .....                      | 347        |
| 11.1.6    | Flash-Anwendungen als Objekt einbinden .....                     | 348        |
| 11.1.7    | Rahmen um Objekte .....                                          | 350        |
| 11.1.8    | Namen für Objekte .....                                          | 352        |
| 11.2      | Eigenschaften von Objekten .....                                 | 352        |
| 11.2.1    | Objekt zum umgebenden Text ausrichten .....                      | 352        |
| 11.2.2    | Text um ein Objekt fließen lassen .....                          | 353        |
| 11.2.3    | Meldungstext während des Ladevorgangs anzeigen .....             | 355        |
| 11.2.4    | Objekte nur bei Aufruf durch Anwender anzeigen .....             | 356        |
| 11.2.5    | Objekte formatieren mit CSS .....                                | 357        |
| 11.3      | Java-Applets einbinden (herkömmlich) .....                       | 357        |
| 11.3.1    | Java-Applets einbinden .....                                     | 357        |
| 11.3.2    | Java-Applets ausrichten .....                                    | 360        |
| 11.4      | Multimedia einbinden (Netscape herkömmlich) .....                | 361        |
| 11.4.1    | Multimedia-Bereich definieren .....                              | 361        |
| 11.4.2    | Multimedia-Bereiche ausrichten .....                             | 363        |
| 11.4.3    | Alternativer Inhalt für Browser .....                            | 363        |
| <b>12</b> | <b>Layer (Netscape) .....</b>                                    | <b>365</b> |
| 12.1      | Layer definieren und positionieren .....                         | 365        |
| 12.1.1    | Allgemeines zu Layern .....                                      | 365        |
| 12.1.2    | Layer definieren .....                                           | 366        |
| 12.1.3    | Inline-Layer definieren .....                                    | 367        |
| 12.1.4    | Anzeigebereich von Layern beschneiden .....                      | 368        |
| 12.1.5    | Layer mit Inhalt aus anderer Datei .....                         | 369        |
| 12.1.6    | Layer verschachteln .....                                        | 370        |
| 12.1.7    | Hintergrundfarbe für Layer .....                                 | 371        |
| 12.1.8    | Hintergrundbild (Wallpaper) für Layer .....                      | 372        |
| 12.2      | Layer anordnen und kontrollieren .....                           | 373        |
| 12.2.1    | Angaben zur Schichtposition mit Layer-Namen .....                | 373        |
| 12.2.2    | Angaben zur Schichtposition mit Indexnummern .....               | 374        |
| 12.2.3    | Layer verstecken und anzeigen .....                              | 375        |
| <b>13</b> | <b>Universalattribute und weiterführende HTML-Elemente .....</b> | <b>377</b> |
| 13.1      | Allgemeine Universalattribute .....                              | 377        |
| 13.2      | Event-Handler für Scripts .....                                  | 379        |
| 13.3      | Weiterführende HTML-Elemente .....                               | 380        |
| 13.3.1    | Stylesheet-Bereiche definieren .....                             | 380        |
| 13.4      | Script-Bereiche in HTML .....                                    | 382        |
| 13.4.1    | Script-Bereich definieren .....                                  | 382        |
| 13.4.2    | Noscript-Bereich definieren .....                                | 384        |
| 13.4.3    | Weitere Angaben zum Script .....                                 | 385        |

|                                                                             |     |
|-----------------------------------------------------------------------------|-----|
| <b>14 XHTML und HTML</b>                                                    | 387 |
| 14.1 Unterschiede zwischen XHTML und HTML                                   | 387 |
| 14.1.1 Allgemeines zu XHTML                                                 | 387 |
| 14.1.2 Unterschied: Mime-Typen                                              | 388 |
| 14.1.3 Unterschied: Dateinamen                                              | 388 |
| 14.1.4 Unterschied: Die XML-Deklaration                                     | 389 |
| 14.1.5 Unterschied: Dokumenttyp-Angaben                                     | 390 |
| 14.1.6 Unterschied: HTML-Wurzelement mit Namensraumangabe                   | 391 |
| 14.1.7 Unterschied: Strengeres Einhalten des HTML-Grundgerüsts              | 391 |
| 14.1.8 Unterschied: Kleinschreibung                                         | 392 |
| 14.1.9 Unterschied: Leere Elemente                                          | 392 |
| 14.1.10 Unterschied: Elemente mit optionalem Abschluss-Tag                  | 393 |
| 14.1.11 Unterschied: Allein stehende (leere) Attribute                      | 393 |
| 14.1.12 Unterschied: Leerraum in Wertzuweisungen                            | 394 |
| 14.1.13 Unterschied: Verweise zu Ankern                                     | 394 |
| 14.1.14 Unterschied: Das Universalattribut lang=                            | 395 |
| 14.1.15 Unterschied: Inhalt von Script- und Style-Bereichen                 | 395 |
| 14.1.16 Unterschied: Ausnahmen von Verschachtelungsregeln                   | 396 |
| 14.1.17 Beispiel eines XHTML-gerechten Dokuments                            | 396 |
| 14.2 Modularisierung von XHTML und XHTML 1.1                                | 398 |
| 14.2.1 Allgemeines zur Modularisierung von XHTML                            | 398 |
| 14.2.2 Die Module von XHTML 1.1                                             | 400 |
| <b>15 CSS Stylesheets</b>                                                   | 403 |
| 15.1 CSS Stylesheets und HTML                                               | 403 |
| 15.1.1 Sinn und Zweck von CSS Stylesheets                                   | 403 |
| 15.1.2 Stylesheet-Sprachen, -Versionen und -Informationen                   | 403 |
| 15.1.3 Stylesheets, Web-Browser und Anwender                                | 404 |
| 15.1.4 CSS-Formate und CSS-Eigenschaften                                    | 406 |
| 15.2 CSS Stylesheets in HTML einbinden                                      | 406 |
| 15.2.1 Formate zentral für eine HTML-Datei definieren                       | 406 |
| 15.2.2 Formate zentral in separater CSS-Datei definieren                    | 407 |
| 15.2.3 Formate innerhalb eines HTML-Elements definieren                     | 408 |
| 15.2.4 Verschiedene separate Stylesheets für unterschiedliche Ausgabemedien | 408 |
| 15.2.5 Formatdefinitionen für unterschiedliche Ausgabemedien                | 410 |
| 15.2.6 Kommentare innerhalb von Stylesheet-Bereichen                        | 411 |
| 15.2.7 Stylesheet-Sprache ausdrücklich festlegen                            | 412 |
| 15.3 Zentrale Formate definieren                                            | 412 |
| 15.3.1 Aufbau zentraler Formate                                             | 412 |
| 15.3.2 Formate für HTML-Elemente definieren                                 | 413 |
| 15.3.3 Formate für verschachtelte HTML-Elemente definieren                  | 415 |
| 15.3.4 Attributbedingte Formate definieren                                  | 417 |
| 15.3.5 Formate für Klassen definieren                                       | 419 |
| 15.3.6 Klassen in Verbindung mit den Elementen div und span                 | 420 |
| 15.3.7 Individualformate definieren                                         | 422 |

|           |                                                     |            |
|-----------|-----------------------------------------------------|------------|
| 15.3.8    | Pseudoformate definieren.....                       | 424        |
| 15.4      | HTML-Elemente direktformatieren.....                | 425        |
| 15.4.1    | Formatdefinitionen mit dem style-Attribut.....      | 425        |
| 15.5      | Maßeinheiten, Farbangaben und Wertzuweisung.....    | 426        |
| 15.5.1    | Numerische Angaben.....                             | 426        |
| 15.5.2    | Farbangaben.....                                    | 428        |
| 15.5.3    | Erlaubte feste Werte und Zeichenkettenangaben.....  | 430        |
| 15.5.4    | Numerische Angaben für Sprachausgabe.....           | 431        |
| <b>16</b> | <b>CSS-Eigenschaften.....</b>                       | <b>433</b> |
| 16.1      | Schriftformatierung.....                            | 433        |
| 16.2      | Schriftformatierung mit Schriftartendatei.....      | 444        |
| 16.3      | Ausrichtung und Absatzkontrolle.....                | 448        |
| 16.4      | Außenrand und Abstand.....                          | 454        |
| 16.5      | Innenabstand.....                                   | 460        |
| 16.6      | Rahmen.....                                         | 463        |
| 16.7      | Hintergrundfarben und -bilder.....                  | 471        |
| 16.8      | Listenformatierung.....                             | 477        |
| 16.8.1    | list-style-type (Darstellungstyp).....              | 478        |
| 16.9      | Tabellenformatierung.....                           | 483        |
| 16.10     | Pseudoformate.....                                  | 489        |
| 16.11     | Positionierung und Anzeige von Elementen.....       | 494        |
| 16.12     | Layouts für Printmedien.....                        | 511        |
| 16.13     | Sprachausgabe.....                                  | 517        |
| 16.14     | Anzeigefenster.....                                 | 526        |
| 16.14.1   | cursor (Mauszeiger).....                            | 527        |
| 16.15     | Filter (nur Microsoft).....                         | 530        |
| <b>17</b> | <b>JavaScript und DOM.....</b>                      | <b>543</b> |
| 17.1      | JavaScript und HTML.....                            | 543        |
| 17.2      | Standards, Sprachvarianten und Sprachversionen..... | 544        |
| 17.3      | Hinweise für die Praxis.....                        | 548        |
| 17.4      | JavaScript-Bereiche in HTML definieren.....         | 549        |
| 17.5      | JavaScript-Anweisungen in HTML-Tags.....            | 550        |
| 17.6      | JavaScript in separaten Dateien.....                | 551        |
| <b>18</b> | <b>JavaScript: Sprachelemente.....</b>              | <b>553</b> |
| 18.1      | Allgemeine Regeln für JavaScript.....               | 553        |
| 18.1.1    | Anweisungen notieren.....                           | 553        |
| 18.1.2    | Anweisungsblöcke notieren.....                      | 554        |
| 18.1.3    | Selbst vergebene Namen.....                         | 555        |
| 18.1.4    | Kommentare in JavaScript.....                       | 556        |
| 18.2      | Variablen und Werte.....                            | 556        |

|         |                                                               |     |
|---------|---------------------------------------------------------------|-----|
| 18.2.1  | Variablen definieren .....                                    | 556 |
| 18.2.2  | Werte von Variablen ändern .....                              | 558 |
| 18.3    | Objekte, Eigenschaften und Methoden.....                      | 559 |
| 18.3.1  | Vordefinierte JavaScript-Objekte .....                        | 559 |
| 18.3.2  | Vordefinierte JavaScript-Objekte verwenden .....              | 560 |
| 18.3.3  | Eigene Objekte definieren .....                               | 561 |
| 18.3.4  | Eigenschaften von Objekten .....                              | 563 |
| 18.3.5  | Objekt-Methoden .....                                         | 564 |
| 18.3.6  | Abfragen, ob ein Objekt existiert .....                       | 564 |
| 18.3.7  | Mehrere Anweisungen mit einem Objekt ausführen (with) .....   | 565 |
| 18.3.8  | Auf aktuelles Objekt Bezug nehmen (this).....                 | 566 |
| 18.4    | Funktionen.....                                               | 567 |
| 18.4.1  | Funktion definieren .....                                     | 567 |
| 18.4.2  | Funktion aufrufen.....                                        | 568 |
| 18.4.3  | Funktion mit Rückgabewert aufrufen .....                      | 569 |
| 18.4.4  | Vordefinierte JavaScript-Funktionen .....                     | 571 |
| 18.5    | Steuerzeichen und besondere Notationen .....                  | 572 |
| 18.5.1  | Steuerzeichen bei Zeichenketten .....                         | 572 |
| 18.5.2  | Notation numerischer Werte .....                              | 572 |
| 18.6    | Operatoren .....                                              | 573 |
| 18.6.1  | Zuweisungsoperator .....                                      | 573 |
| 18.6.2  | Vergleichsoperatoren .....                                    | 573 |
| 18.6.3  | Berechnungsoperatoren .....                                   | 574 |
| 18.6.4  | Logische Operatoren.....                                      | 575 |
| 18.6.5  | Bit-Operatoren .....                                          | 576 |
| 18.6.6  | Operator zur Zeichenkettenverknüpfung .....                   | 576 |
| 18.6.7  | Operatorenrangfolge .....                                     | 576 |
| 18.6.8  | Operator zur Typenbestimmung .....                            | 577 |
| 18.6.9  | void-Operator .....                                           | 578 |
| 18.6.10 | Operator zum Löschen von Objekten .....                       | 578 |
| 18.7    | Bedingte Anweisungen (if-else/switch) .....                   | 578 |
| 18.7.1  | Wenn-dann-Bedingungen mit "if" .....                          | 578 |
| 18.7.2  | Einfache Entweder-oder-Abfrage .....                          | 579 |
| 18.7.3  | Fallunterscheidung mit »switch«.....                          | 581 |
| 18.8    | Schleifen (while/for/do-while) .....                          | 582 |
| 18.8.1  | Schleifen mit »while« .....                                   | 582 |
| 18.8.2  | Schleifen mit »for« .....                                     | 583 |
| 18.8.3  | Kontrolle innerhalb von Schleifen – break und continue.....   | 586 |
| 18.9    | Reservierte Wörter.....                                       | 587 |
| 18.9.1  | Übersicht reservierter Wörter .....                           | 587 |
| 18.10   | Event-Handler .....                                           | 589 |
| 18.10.1 | onMouseover (beim Überfahren des Elements mit der Maus) ..... | 600 |

|           |                                               |     |
|-----------|-----------------------------------------------|-----|
| <b>19</b> | <b>JavaScript: Beispiele</b>                  | 605 |
| 19.1      | Zwei Frames gleichzeitig ändern               | 605 |
| 19.2      | Seitenanzeige in Frames verhindern            | 609 |
| 19.3      | Taschenrechner                                | 612 |
| 19.4      | Monatskalender                                | 617 |
| 19.5      | Persönliche Seitenbesuche zählen mit Cookies  | 622 |
| 19.6      | Formulareingaben überprüfen                   | 625 |
| 19.7      | Dynamische grafische Buttons                  | 628 |
| 19.8      | Aufenthaltsdauer auf Web-Seite anzeigen       | 632 |
| 19.9      | Verweisliste als Auswahlliste                 | 634 |
| 19.10     | Fehlerbehandlung mit onError                  | 638 |
| 19.11     | Fehlerbehandlung mit dem Statement try..catch | 640 |
| 19.12     | Änderungen der Fenstergröße überwachen        | 642 |
| <b>20</b> | <b>JavaScript: Objekt-Referenz</b>            | 647 |
| 20.1      | Hinweise zur Objekt-Referenz                  | 648 |
| 20.1.1    | Objekthierarchie                              | 648 |
| 20.1.2    | Objekt-Referenz verwenden                     | 648 |
| 20.2      | window                                        | 649 |
| 20.3      | frames                                        | 677 |
| 20.4      | document                                      | 681 |
| 20.5      | HTML-Elementobjekte                           | 700 |
| 20.6      | node                                          | 775 |
| 20.7      | all                                           | 796 |
| 20.8      | style                                         | 815 |
| 20.9      | anchors                                       | 821 |
| 20.10     | applets                                       | 824 |
| 20.11     | forms                                         | 826 |
| 20.12     | elements                                      | 833 |
| 20.13     | options                                       | 842 |
| 20.14     | images                                        | 850 |
| 20.15     | embeds                                        | 857 |
| 20.16     | layers                                        | 862 |
| 20.17     | links                                         | 879 |
| 20.18     | event                                         | 883 |
| 20.19     | history                                       | 891 |
| 20.20     | location                                      | 893 |
| 20.21     | Array                                         | 897 |
| 20.22     | Boolean                                       | 908 |
| 20.23     | Date                                          | 909 |
| 20.24     | Function                                      | 926 |
| 20.25     | Math                                          | 929 |

|           |                                                               |            |
|-----------|---------------------------------------------------------------|------------|
| 20.26     | navigator .....                                               | 939        |
| 20.27     | mimeType .....                                                | 943        |
| 20.28     | plugins.....                                                  | 946        |
| 20.29     | Number .....                                                  | 949        |
| 20.30     | RegExp .....                                                  | 955        |
| 20.31     | Screen .....                                                  | 959        |
| 20.32     | string .....                                                  | 962        |
| 20.33     | Objektunabhängige Funktionen .....                            | 977        |
| <b>21</b> | <b>Dynamisches HTML .....</b>                                 | <b>985</b> |
| 21.1      | Allgemeines zu Dynamischem HTML.....                          | 985        |
| 21.1.1    | Das »Dynamische« am Dynamischen HTML .....                    | 985        |
| 21.1.2    | Das Problem der inkompatiblen Modelle .....                   | 986        |
| 21.2      | Das Document Object Model (DOM).....                          | 986        |
| 21.2.1    | Allgemeines zum DOM .....                                     | 986        |
| 21.2.2    | Elementinhalte dynamisch ändern.....                          | 988        |
| 21.2.3    | HTML-Attribute dynamisch ändern.....                          | 991        |
| 21.2.4    | CSS-Eigenschaften dynamisch ändern .....                      | 993        |
| 21.2.5    | Neue Knoten erzeugen und in Baumstruktur einhängen .....      | 995        |
| 21.2.6    | Event-Handling beim DOM .....                                 | 998        |
| 21.3      | Das ältere DHTML-Modell von Microsoft .....                   | 1001       |
| 21.3.1    | Allgemeines zum Microsoft-Modell.....                         | 1001       |
| 21.3.2    | Script-Sprachen: JavaScript, JScript und VBScript .....       | 1002       |
| 21.3.3    | Das all-Objekt .....                                          | 1003       |
| 21.3.4    | CSS Stylesheets und Scripts .....                             | 1005       |
| 21.3.5    | Schreibweise von CSS-Eigenschaften in JavaScript/JScript..... | 1007       |
| 21.3.6    | Event-Handler und das Event-Objekt .....                      | 1009       |
| 21.3.7    | Vererbung bei Events (Event-Bubbling).....                    | 1010       |
| 21.4      | Das ältere DHTML-Modell von Netscape .....                    | 1012       |
| 21.4.1    | Allgemeines zum DHTML-Modell von Netscape 4.x.....            | 1012       |
| 21.4.2    | Layer-Bereiche dynamisch anzeigen/verstecken .....            | 1013       |
| 21.4.3    | Layer-Bereiche dynamisch überlappen.....                      | 1015       |
| 21.4.4    | Layer-Bereiche dynamisch bewegen .....                        | 1018       |
| 21.4.5    | Layer-Bereiche dynamisch vergrößern/verkleinern .....         | 1022       |
| 21.4.6    | Layer-Inhalte dynamisch austauschen.....                      | 1025       |
| 21.4.7    | Layer ohne layer-Element simulieren.....                      | 1027       |
| 21.4.8    | Das Event-Objekt .....                                        | 1029       |
| 21.4.9    | Inline-Scripts bei Layern .....                               | 1032       |
| 21.5      | Datenanbindung (nur Microsoft).....                           | 1034       |
| 21.5.1    | Allgemeines zur Datenanbindung.....                           | 1034       |
| 21.5.2    | Beispiel: Datei dynamisch in Tabelle einbinden .....          | 1035       |
| 21.5.3    | Beispiel: Daten durchsuchbar machen .....                     | 1038       |
| 21.5.4    | HTML-Elemente für die Anzeige von externen Daten .....        | 1043       |
| 21.5.5    | Datenmodell: Tabular Data Control (TDC).....                  | 1048       |

|           |                                                        |             |
|-----------|--------------------------------------------------------|-------------|
| 21.5.6    | Datenmodell: Remote Data Service (RDS).....            | 1050        |
| 21.5.7    | Datenmodell: JDBC DataSource Applet.....               | 1051        |
| 21.5.8    | Datenmodell: XML Data Source .....                     | 1053        |
| 21.5.9    | Datenmodell: MSHTML Data Source .....                  | 1054        |
| 21.6      | Dynamische Filter (nur Microsoft).....                 | 1054        |
| 21.6.1    | Allgemeines zu dynamischen Filtern .....               | 1054        |
| 21.6.2    | Einfaches Dynamisches Ändern von Filterwerten.....     | 1055        |
| 21.6.3    | Fließende Übergänge mit dem blendTrans-Filter.....     | 1057        |
| 21.6.4    | Fließende Übergänge mit dem revealTrans-Filter .....   | 1060        |
| 21.6.5    | Ein-/Ausblend-Effekte .....                            | 1062        |
| 21.6.6    | Fließende Seitenübergänge .....                        | 1065        |
| <b>22</b> | <b>DHTML – Praxis-Lösungen .....</b>                   | <b>1067</b> |
| 22.1      | Allgemeine DHTML-Bibliothek.....                       | 1067        |
| 22.1.1    | Quelltext mit Erläuterungen .....                      | 1067        |
| 22.1.2    | Test der DHTML-Bibliothek .....                        | 1075        |
| 22.1.3    | DHTML-Bibliothek erweitern .....                       | 1078        |
| 22.2      | Dynamische Navigationsleiste .....                     | 1078        |
| 22.2.1    | Hinweise zu diesem Beispiel .....                      | 1078        |
| 22.2.2    | Quelltext mit Erläuterungen .....                      | 1079        |
| 22.3      | Laufende Datums-/Uhrzeitanzeige.....                   | 1084        |
| 22.3.1    | Hinweise zu diesem Beispiel .....                      | 1084        |
| 22.3.2    | Quelltext mit Erläuterungen .....                      | 1085        |
| 22.4      | Bilderbuch zum Umblättern .....                        | 1087        |
| 22.4.1    | Hinweise zu diesem Beispiel .....                      | 1087        |
| 22.4.2    | Quelltext mit Erläuterungen .....                      | 1088        |
| <b>23</b> | <b>Internationalisierung .....</b>                     | <b>1093</b> |
| 23.1      | Computer und geschriebene Sprache.....                 | 1093        |
| 23.1.1    | Bits, Bytes und Zeichen .....                          | 1093        |
| 23.1.2    | Beispiel für Zeichensätze .....                        | 1094        |
| 23.1.3    | Die Grenzen von Bytes und Zeichensätzen sprengen ..... | 1095        |
| 23.1.4    | Die Bedeutung der Schriftarten .....                   | 1095        |
| 23.1.5    | Schriftkulturen mit anderer Schreibrichtung .....      | 1096        |
| 23.2      | Zeichensätze (ISO-8859-Familie und andere).....        | 1096        |
| 23.2.1    | Die ISO-8859-Zeichensätze .....                        | 1096        |
| 23.2.2    | Liste der ISO-8859-Zeichensätze.....                   | 1098        |
| 23.2.3    | Alle offiziellen Zeichensätze .....                    | 1101        |
| 23.3      | Das Unicode-System .....                               | 1102        |
| 23.3.1    | Allgemeines zu Unicode .....                           | 1102        |
| 23.3.2    | Aufbau des Unicode-Systems.....                        | 1103        |
| 23.4      | Downloadbare Schriftarten.....                         | 1105        |
| 23.4.1    | Allgemeines zu downloadbaren Schriftarten .....        | 1105        |
| 23.4.2    | TrueDoc (Bitstream).....                               | 1106        |



|           |                                                             |             |
|-----------|-------------------------------------------------------------|-------------|
| 23.4.3    | Microsoft und das Web Embedding Fonts Tool WEFT .....       | 1107        |
| 23.4.4    | Downloadbare Schriftarten in HTML einbinden .....           | 1108        |
| <b>24</b> | <b>Grafik</b> .....                                         | <b>1111</b> |
| 24.1      | Grafikformate für Web-Seiten .....                          | 1111        |
| 24.1.1    | Allgemeines zu Grafikformaten für Web-Seiten .....          | 1111        |
| 24.1.2    | Das GIF-Format .....                                        | 1111        |
| 24.1.3    | Das JPEG-Format .....                                       | 1112        |
| 24.1.4    | Das PNG-Format .....                                        | 1113        |
| 24.1.5    | Das LuraWave-Format .....                                   | 1114        |
| 24.1.6    | Das SVG-Format.....                                         | 1115        |
| 24.2      | Typische Grafiksorten für Web-Seiten .....                  | 1116        |
| 24.2.1    | Buttons (Schaltflächen) .....                               | 1116        |
| 24.2.2    | Dots (Auflistungspunkte) .....                              | 1117        |
| 24.2.3    | Bars (Trennleisten) .....                                   | 1118        |
| 24.2.4    | Wallpapers (Hintergrundbilder) .....                        | 1119        |
| 24.2.5    | Symbole (Textergänzung oder Textersatz).....                | 1121        |
| 24.2.6    | Cliparts (Illustrationen) .....                             | 1121        |
| 24.2.7    | Thumbnails (Vorschaugrafiken) .....                         | 1123        |
| 24.2.8    | Fotos.....                                                  | 1124        |
| 24.2.9    | Blinde Pixel .....                                          | 1124        |
| 24.2.10   | Grafische Layout-Elemente .....                             | 1125        |
| 24.3      | Techniken und Begriffe der Bildbearbeitung .....            | 1126        |
| 24.3.1    | Allgemeines zu Raster- bzw. Pixelgrafiken .....             | 1126        |
| 24.3.2    | Bildgröße, dpi und Pixel .....                              | 1127        |
| 24.3.3    | Farbtiefen, Farbmodelle und Farbkanäle .....                | 1128        |
| 24.3.4    | Farbpaletten und indizierte Farben .....                    | 1129        |
| 24.3.5    | Alphakanal und echte Transparenz .....                      | 1129        |
| 24.3.6    | Transparenz einer bestimmten Palettenfarbe .....            | 1130        |
| 24.3.7    | Dithering (Error-Diffusion) .....                           | 1130        |
| 24.3.8    | Graustufen .....                                            | 1132        |
| 24.3.9    | Helligkeit, Sättigung und Kontrast .....                    | 1132        |
| 24.3.10   | Histogramm.....                                             | 1134        |
| 24.3.11   | Gammakorrektur.....                                         | 1135        |
| 24.3.12   | Filtereffekte .....                                         | 1136        |
| 24.3.13   | Anti-Aliasing.....                                          | 1139        |
| 24.4      | Animierte GIF-Grafiken .....                                | 1140        |
| 24.4.1    | Allgemeines zu animierten GIF-Grafiken .....                | 1140        |
| 24.4.2    | Diashow-Effekte durch animierte GIF-Grafiken .....          | 1140        |
| 24.4.3    | Daumenkino-Effekte durch animierte GIF-Grafiken .....       | 1141        |
| 24.4.4    | Vorgehensweise beim Erstellen animierter GIF-Grafiken ..... | 1142        |
| <b>25</b> | <b>Web-Projektverwaltung</b> .....                          | <b>1143</b> |
| 25.1      | Web-Projekte planen .....                                   | 1143        |
| 25.1.1    | Allgemeines zu Web-Projekten .....                          | 1143        |

|          |                                                             |             |
|----------|-------------------------------------------------------------|-------------|
| 25.1.2   | Ziel des Projekts .....                                     | 1143        |
| 25.1.3   | Zielgruppe des Projekts .....                               | 1144        |
| 25.1.4   | Projektumgebung im Web erforschen .....                     | 1145        |
| 25.1.5   | Inhalte und Redaktion .....                                 | 1145        |
| 25.1.6   | Kostenaufwand .....                                         | 1146        |
| 25.1.7   | Terminierung und Ankündigung .....                          | 1147        |
| 25.1.8   | Gliederung und Strukturierung.....                          | 1148        |
| 25.1.9   | Corporate Design.....                                       | 1148        |
| 25.1.10  | Web-Design .....                                            | 1149        |
| 25.2     | Web-Projekte publizieren.....                               | 1150        |
| 25.2.1   | Allgemeines zum Publizieren von Web-Projekten .....         | 1150        |
| 25.2.2   | Homepages bei Online-Diensten .....                         | 1151        |
| 25.2.3   | Kostenlose Homepages .....                                  | 1152        |
| 25.2.4   | Eigene Domain-Adresse.....                                  | 1152        |
| 25.2.5   | Virtual Hosting mit eigener Domain .....                    | 1152        |
| 25.2.6   | Server-Hosting mit eigener Domain.....                      | 1153        |
| 25.2.7   | Server-Housing mit eigener Domain .....                     | 1154        |
| 25.2.8   | Publizieren mit FTP.....                                    | 1154        |
| 25.3     | Web-Projekte bekannt machen.....                            | 1155        |
| 25.3.1   | Allgemeines .....                                           | 1155        |
| 25.3.2   | Projekte bei Suchdiensten und Verzeichnissen anmelden ..... | 1155        |
| 25.3.3   | Strategien bei der Online-Kommunikation .....               | 1158        |
| 25.3.4   | Sonstige Bekanntmachungsstrategien .....                    | 1158        |
| 25.4     | Web-Projekte kontrollieren .....                            | 1159        |
| 25.4.1   | Allgemeines .....                                           | 1159        |
| 25.4.2   | Log-Dateien des Web-Servers .....                           | 1160        |
| 25.4.3   | Statistiken .....                                           | 1161        |
| 25.4.4   | Statistiken auswerten.....                                  | 1162        |
| 25.4.5   | Begriffe wie Hits, Pageviews und Visits.....                | 1163        |
| 25.4.6   | Anwender-Feedback auswerten .....                           | 1164        |
| 25.5     | Provider wechseln mit Web-Projekten.....                    | 1165        |
| 25.5.1   | Provider wechseln bei vorhandener Domain-Adresse .....      | 1165        |
| 25.5.2   | Provider wechseln und neue Domain-Adresse.....              | 1165        |
| 25.6     | Rechtliche Aspekte von Web-Projekten .....                  | 1166        |
| 25.6.1   | Allgemeines .....                                           | 1166        |
| 25.6.2   | Hoheit und Verantwortlichkeit.....                          | 1167        |
| 25.6.3   | Relevante Rechtsbereiche (Deutschland).....                 | 1168        |
| 25.6.4   | Gesetze für »neue Medien« (in Deutschland).....             | 1169        |
| 25.6.5   | Autoren-Aussage zur Link-Haftung.....                       | 1171        |
| <b>A</b> | <b>Technische Ergänzungen .....</b>                         | <b>1173</b> |
| A.1      | HTTP-Statuscodes .....                                      | 1173        |
| A.2      | Mime-Typen .....                                            | 1178        |
| A.3      | Sprachenkürzel .....                                        | 1183        |

|          |                                                                   |             |
|----------|-------------------------------------------------------------------|-------------|
| A.4      | Client-Identifikationen .....                                     | 1189        |
| A.5      | Client-Identifikationen anderer Browser .....                     | 1191        |
| A.6      | robots.txt – Robots kontrollieren .....                           | 1193        |
| A.7      | .htaccess – Server-Reaktionen kontrollieren .....                 | 1195        |
| <b>B</b> | <b>HTML: Elementreferenz .....</b>                                | <b>1207</b> |
| B.1      | Allgemeines zur HTML-Elementreferenz .....                        | 1207        |
| B.2      | Block-Elemente .....                                              | 1207        |
| B.3      | Inline-Elemente .....                                             | 1207        |
| B.4      | #PCDATA .....                                                     | 1208        |
| B.5      | Elementreferenz .....                                             | 1208        |
| <b>C</b> | <b>HTML: Attributreferenz .....</b>                               | <b>1225</b> |
| C.1      | Allgemeines zur HTML-Attributreferenz .....                       | 1225        |
| C.2      | #CDATA .....                                                      | 1225        |
| C.3      | ID, IDREF oder Name .....                                         | 1225        |
| C.4      | Zahl .....                                                        | 1226        |
| C.5      | Universalattribute .....                                          | 1226        |
| C.6      | Attributreferenz .....                                            | 1228        |
| <b>D</b> | <b>HTML: Zeichenreferenz .....</b>                                | <b>1263</b> |
| D.1      | Allgemeines zur HTML-Zeichenreferenz .....                        | 1263        |
| D.2      | Benannte Zeichen für HTML-eigene Zeichen .....                    | 1264        |
| D.3      | Benannte Zeichen für den Zeichensatz ISO 8859-1 .....             | 1264        |
| D.4      | Benannte Zeichen für griechische Buchstaben .....                 | 1266        |
| D.5      | Benannte Zeichen für mathematische Symbole .....                  | 1268        |
| D.6      | Benannte Zeichen für technische Symbole .....                     | 1269        |
| D.7      | Benannte Zeichen für Pfeil-Symbole .....                          | 1269        |
| D.8      | Benannte Zeichen für diverse Symbole .....                        | 1270        |
| D.9      | Benannte Zeichen für Interpunktion .....                          | 1270        |
| <b>E</b> | <b>HTML: Varianten .....</b>                                      | <b>1273</b> |
| E.1      | Varianten »strict«, »transitional« und »frameset« .....           | 1273        |
| E.1.1    | Variante »strict« .....                                           | 1273        |
| E.1.2    | Variante »transitional« .....                                     | 1273        |
| E.1.3    | Variante »frameset« .....                                         | 1274        |
| E.2      | Nicht erlaubte Elemente und Attribute bei Variante »strict« ..... | 1274        |
|          | <b>Stichwortverzeichnis .....</b>                                 | <b>1279</b> |

# 1 Internet und WWW

## 1.1 Entstehung des Internet

### 1.1.1 Das ARPA-Net

Die Ursprünge des heutigen Internet reichen in die 60er Jahre zurück. Es war die Zeit des Kalten Krieges zwischen den beiden Weltmächten USA und UdSSR. Neue Impulse in der Elektronischen Datenverarbeitung (EDV) kamen in jener Zeit hauptsächlich durch militärische Initiativen zustande. Mittlerweile gibt es zwar im Internet Proteste gegen die Auffassung, militärische Interessen hätten das Internet geboren. Das ist insofern berechtigt, als es keine unmittelbare Ursachen-Wirkungs-Verhältnisse gibt. Doch die technologischen Ideen und Entwicklungen der Vorläufernetze kamen definitiv aus dem militärnahen Umfeld in den USA, und es ist deshalb auch nicht verkehrt, das so darzustellen.

Im Department of Defense, dem amerikanischen Verteidigungsministerium, wurde seit den frühen 60er Jahren überlegt, wie man wichtige militärische Daten besser schützen könnte. Selbst bei einem atomaren Angriff des Gegners sollten die Daten nicht zerstört werden können. Als Lösung kam nur ein elektronisches Datennetz in Frage. Die gleichen Daten sollten dabei auf mehreren, weit voneinander entfernten Rechnern abgelegt werden. Bei neuen oder geänderten Daten sollten sich alle angeschlossenen Rechner binnen kürzester Zeit den aktuellen Datenstand zusenden. Jeder Rechner sollte dabei über mehrere Wege mit jedem anderen Rechner kommunizieren können. So würde das Netz auch dann funktionieren, wenn ein einzelner Rechner oder eine bestimmte Leitung durch einen Angriff zerstört würde. So gab die US Air Force 1964 bei der RAND Corporation das so genannte »dezentrale Netzwerk« in Auftrag. Bei dem, was da entwickelt werden sollte, handelt es sich noch nicht um den direkten Vorläufer des heutigen Internet. Nach vielem Hin und Her scheiterte dieses Projekt auch und wurde nie realisiert. Die Idee des »dezentralen Netzwerks« mit der paketweisen Datenübertragung blieb jedoch in den Köpfen hängen.

Die Advanced Research Projects Agency (ARPA), eine seit 1958 bestehende wissenschaftliche Einrichtung, deren Forschungsergebnisse in militärische Zwecke einfließen, entschloss sich 1966 zur Vernetzung der ARPA-eigenen Großrechner. Dabei wurde die Idee des »dezentralen Netzwerks« wieder aufgegriffen. Ende 1969 waren die ersten vier Rechner an das ARPA-Net angeschlossen. Drei Jahre später waren es bereits 40 Rechner. In dieser Zeit war es jedoch das ARPA-eigene Netz. In den ersten Jahren wurde das Netz deshalb auch **ARPA-Net** genannt. Aus ihm sollte später das Internet erwachsen.

Die Geschichte des Internet beginnt also zur Zeit der ersten Mondlandung und in jener Zeit, als die Hippies das Zeitalter des Wassermanns heraufziehen sahen. Informationen zum ARPA-Net im WWW:

- Defense Advanced Research Projects Agency (DARPA) ([www.arpa.mil/](http://www.arpa.mil/))

### 1.1.2 Wissenschaftliche Einrichtungen

Das Prinzip der vernetzten Rechner war aber nicht nur für militärische Zwecke interessant. Man erkannte schnell, dass auch der akademische Betrieb vom ARPA-Net profitieren würde. Für Wissenschaftler war allerdings weniger das Synchronisieren von gleichen Daten auf mehreren Rechnern interessant als vielmehr die Möglichkeit, Daten von einem anderen Rechner abzurufen. Wegen der offenen Architektur des ARPA-Net stand einer solchen Verwendung nichts im Wege. Wissenschaftler konnten von den frühen 70er Jahren an Forschungsergebnisse anderer Institute über das ARPA-Net abrufen oder anderen angeschlossenen Instituten eigene Daten zur Verfügung stellen.

Die Anzahl der angeschlossenen Rechner stieg an. Es handelte sich um sehr unterschiedliche Rechnerarten mit nicht kompatiblen Betriebssystemen und unterschiedlichem Netzzugang. Großrechner verschiedener Fabrikate, Unix-Rechner und später auch Personalcomputer drängten ins Netz. Einige hatten eine Standleitung, also eine ständige Internet-Verbindung, andere wählten sich über Telefon und Modem ein. Um die unterschiedlichen Voraussetzungen unter einen Hut zu bringen, entstand die Notwendigkeit, ein neues Datenübertragungsprotokoll für das Netz zu entwickeln. Das Protokoll sollte nicht an bestimmte Computersysteme, Übertragungswege oder Übertragungsgeschwindigkeiten gebunden sein. Aus den Bemühungen um ein solches Protokoll ging schließlich das TCP/IP-Protokoll hervor. Datenübertragungen im ARPA-Net liefen nach Einführung von TCP/IP nach einem einheitlichen und standardisierten Schema ab.

Zu den wissenschaftlichen Einrichtungen gehörten natürlich auch Studenten. Die entdeckten das Netz auf ihre Weise. Eine Art »black board« war ihr Wunsch, ein Nachrichtenbrett wie in Universitäten üblich, für Mitfahrgelegenheiten, Jobs, Wohnmöglichkeiten, Reisepartner oder die Möglichkeit, einfach nur zu diskutieren und zu labern. So entstand das Usenet, die Hauptader der heutigen Newsgroups. Viele eingefleischte »Internetter« halten die Newsgroups auch heute noch für den spannendsten und lebendigsten Teil des Internet.

### 1.1.3 Das Netz der Netze

Der Anschluss der akademischen Welt ans Netz erforderte eine Trennung zwischen militärischem und zivilem Teil, da die Militärs ihre eigenen Interessen wahren wollten. So kam es, dass Anfang der 80er Jahre ein neues militärisches Datennetz, das Milnet, vom ARPA-Net abgekoppelt wurde. Das ARPA-Net selbst wurde dem wissenschaftlichen Betrieb überlassen.

In diesem zivilen Teil des Netzes nahm die Anzahl der angeschlossenen Rechner im Laufe der 80er Jahre sprunghaft zu. Eine wichtige Rolle spielte dabei die amerikanische **National Science Foundation (NSF)**. Diese Organisation schaffte ein Leitungsverbandsystem, das alle bedeutenden wissenschaftlichen Rechenzentren des Landes miteinander verband. Einzelne Universitätsrechner oder kleinere Rechnernetze konnten sich mit einem Rechenzentrum verbinden und da-

rüber in andere Netze gelangen. So entstand buchstäblich ein Netz der Netze. Als bald bürgerte sich denn auch der Name »Internet« dafür ein. Die Bezeichnung »ARPA-Net« wurde Ende der 80er Jahre verworfen. Das Leitungs-Verbundsystem, über das die kleineren Einzelnetze zu einem Gesamtnetz wurden, erhielt die treffende Bezeichnung **Backbone** (Rückgrat).

In Europa gab es zeitgleich ähnliche Entwicklungen. Auf dem alten Kontinent setzte man jedoch zunächst auf **ISO-Normen**. Von dem amerikanischen TCP/IP-Modell, das nicht ISO-normiert war, wollte man deshalb nichts wissen. 1986 wurde die Organisation **RARE** (**Réseaux Associés pour la Recherche Européenne**) gegründet. Diese Organisation sollte alle Initiativen zur systemübergreifenden Rechnervernetzung europaweit koordinieren. RARE rief dazu ein Projekt namens **COSINE** (**Cooperation for an Open Systems Interconnection Networking in Europe**) ins Leben.

Der Siegeszug des TCP/IP-Protokolls war jedoch nicht mehr aufzuhalten. Unter dem Druck des Erfolgs in Amerika entstand schließlich ein europäisches Datennetz, das multiprotokollfähig war und unter anderem TCP/IP unterstützte. Dieses Netz lief zunächst unter der Bezeichnung **EuropanET**. Verschiedene nationale wissenschaftliche Netzwerke, etwa das Deutsche Forschungsnetz (DFN), wurden daran angeschlossen.

Mittlerweile sorgen transatlantische Leitungen für die Anbindung Europas an den Backbone in den USA. Die Koordination des Internetverkehrs innerhalb Europas wurde der Organisation **RIPE** (**Réseaux IP Européens**) übertragen. Auch in Europa gibt es seit 1992 ein Leitungs-Verbundsystem. Dieses System nennt sich **Ebone** (**Europäischer Internet-Backbone**). Auf anderen Kontinenten gab und gibt es vergleichbare Entwicklungen.

Was wir also heute unter »Internet« verstehen, ist nicht ein einziges homogenes Netz, sondern ein Verbund aus vielen kleinen, territorial oder organisatorisch begrenzten Netzen. Diese Netze besitzen eine Anbindung an die Backbones und damit an das Gesamtnetz. Auch kommerzielle Internet-Provider hängen an entsprechenden Netzen. Und Endanwender, die ihren Internetzugang in Form einer Verbindung zu einem solchen Provider haben, können über diese Brücke das gesamte Internet nutzen. Ein paar Adressen im WWW zu diesen Themen:

- National Science Foundation (NSF, [www.nsf.gov/](http://www.nsf.gov/))
- Deutsches Forschungsnetz (DFN, [www.dfn.de/](http://www.dfn.de/))
- Ebone-Net ([www.ebone.com/](http://www.ebone.com/))
- Réseaux IP Européens (RIPE, [www.ripe.net/](http://www.ripe.net/))

## 1.2 Basis-Standards im Internet

### 1.2.1 Das TCP/IP-Protokoll

TCP/IP ist der kleinste gemeinsame Nenner des gesamten Datenverkehrs im Internet. Erst durch dieses Protokoll wurde historisch gesehen aus einem begrenzten Netz ein Netz der Netze. Egal, ob Sie Web-Seiten aufrufen, E-Mails versenden, mit FTP Dateien downloaden oder mit Telnet auf einem entfernten Rechner arbeiten: Stets werden die Daten auf gleiche Weise adressiert und

transportiert. TCP bedeutet **Transmission Control Protocol** (Protokoll für Übertragungskontrolle), IP bedeutet **Internet Protocol**.

Wenn Sie eine E-Mail verschicken oder eine HTML-Datei im Web aufrufen, werden die Daten bei der Übertragung im Netz in kleine Pakete zerstückelt. Jedes Paket enthält eine Angabe dazu, an welche Adresse es geschickt werden soll und das wievielte Paket innerhalb der Sendung es ist.

Die Adressierung besorgt das IP. Dazu gibt es ein Adressierungsschema, die so genannten IP-Adressen. Dass die Datenpakete auch wirklich beim Empfänger ankommen, und zwar in der richtigen Reihenfolge, dafür sorgt das TCP. Das TCP verwendet Sequenznummern für die einzelnen Pakete einer Sendung. Erst wenn alle Pakete einer Sendung vollständig beim Empfänger angekommen sind, gilt die Übertragung der Daten als abgeschlossen.

Jeder Rechner, der am Internet teilnimmt, ist mit einer IP-Adresse im Netz angemeldet. Rechner, die ans Internet angeschlossen sind, werden als **Hosts** oder **Host-Rechner** bezeichnet. Wenn Sie also mit Ihrem PC im Web surfen oder neue E-Mails abholen, sind Sie mit einer IP-Adresse im Internet angemeldet. Ihr Zugangs-Provider, über dessen Host-Rechner Sie sich einwählen, kann feste IP-Adressen für Sie einrichten. Große Zugangs-Provider, etwa Online-Dienste wie T-Online oder AOL, vergeben auch personenunabhängig dynamische IP-Adressen für jede Internet-Einwahl. Damit ein Rechner am Internet teilnehmen kann, muss er über eine Software verfügen, die das TCP/IP-Protokoll unterstützt. Unter MS Windows ist das beispielsweise die Datei *winsock.dll* im Windows-Verzeichnis.

### 1.2.2 IP-Adressierung

Eine typische IP-Adresse sieht in Dezimalschreibweise so aus: 149.174.211.5 – vier Zahlen also, getrennt durch Punkte. Die Punkte haben die Aufgabe, über- und untergeordnete Netze anzusprechen. So wie zu einer Telefonnummer im weltweiten Telefonnetz eine Landeskennzahl, eine Ortsnetzkenzahl, eine Teilnehmerrufnummer und manchmal auch noch eine Durchwahlnummer gehört, gibt es auch im Internet eine Vorwahl – die **Netzwerknummer**, und eine Durchwahl – die **Hostnummer**.

Der erste Teil einer IP-Adresse ist die Netzwerknummer, der zweite Teil die Hostnummer. Wo die Grenze zwischen Netzwerknummer und Hostnummer liegt, bestimmt ein Klassifizierungsschema für Netztypen. Die folgende Tabelle verdeutlicht dieses Schema. In den Spalten für die IP-Adressierung und einem typischen Beispiel ist die Netzwerknummer (der Vorwahlteil) fett dargestellt. Der Rest der IP-Adresse ist die Hostnummer eines Rechners innerhalb dieses Netzes.

| Netztyp       | Schema          | Typische IP-Adresse |
|---------------|-----------------|---------------------|
| Klasse-A-Netz | xxx.xxx.xxx.xxx | 103.234.123.87      |
| Klasse-B-Netz | xxx.xxx.xxx.xxx | 151.170.102.15      |
| Klasse-C-Netz | xxx.xxx.xxx.xxx | 196.23.155.113      |

Die oberste Hierarchiestufe bilden die so genannten **Klasse-A-Netze**. Nur die erste Zahl einer IP-Adresse ist darin die Netzwerknummer, alle anderen Zahlen sind Hostnummern innerhalb des Netzwerks. Bei Netzwerknummern solcher Netze sind Zahlen zwischen 1 und 126 möglich, d.h., es kann weltweit nur 126 Klasse-A-Netze geben. Eine IP-Adresse, die zu einem Klasse-A-Netz

gehört, ist also daran erkennbar, dass die erste Zahl zwischen 1 und 126 liegt. Das amerikanische Militärnetz ist beispielsweise so ein Klasse-A-Netz. Innerhalb eines Klasse-A-Netzes kann der entsprechende Netzbetreiber die zweite, dritte und vierte Zahl der einzelnen IP-Adressen seiner Netzteilnehmer frei vergeben. Da alle drei Zahlen Werte von 0 bis 255 haben können, kann ein Klasse-A-Netzbetreiber also bis zu 16,7 Millionen IP-Adressen an Host-Rechner innerhalb seines Netzes vergeben.

Die zweithöchste Hierarchiestufe sind die **Klasse-B-Netze**. Die Netzwerknummer solcher Netze erstreckt sich über die beiden ersten Zahlen der IP-Adresse. Bei der ersten Zahl können Klasse-B-Netze Werte zwischen 128 und 191 haben. Eine IP-Adresse, die zu einem Klasse-B-Netz gehört, ist also daran erkennbar, dass die erste Zahl zwischen 128 und 191 liegt. Bei der zweiten Zahl sind Werte zwischen 0 und 255 erlaubt. Dadurch sind etwa 16.000 solcher Netze möglich. Da die Zahlen drei und vier in solchen Netzen ebenfalls Werte zwischen 0 und 255 haben dürfen, können an jedem Klasse-B-Netz bis zu ca. 65.000 Host-Rechner angeschlossen werden. Klasse-B-Netze werden vor allem an große Firmen, Universitäten und Online-Dienste vergeben.

Die unterste Hierarchie stellen die **Klasse-C-Netze** dar. Die erste Zahl einer IP-Adresse eines Klasse-C-Netzes liegt zwischen 192 und 223. Die Zahlen zwei und drei gehören ebenfalls noch zur Netzwerknummer. Über zwei Millionen solcher Netze sind dadurch adressierbar. Vor allem an kleine und mittlere Unternehmen mit direkter Internet-Verbindung, aber auch an kleinere Internet-Provider, werden solche Adressen vergeben. Da nur noch eine Zahl mit Werten zwischen 0 und 255 übrig bleibt, können in einem C-Netz maximal 255 Host-Rechner angeschlossen werden. Eine Zahl davon ist reserviert, also bleiben 254 mögliche Host-Rechner übrig.

Durch die Vergabe dynamischer IP-Adressen pro Einwahl können Netzbetreiber die Anzahl der tatsächlich internetfähigen Anschlüsse deutlich höher halten, als wenn wirklich nur jeder Rechner eine feste Adresse erhalten würde, egal, ob er online ist oder nicht. Ob dieses Adressierungsschema jedoch den Anforderungen der Zukunft noch gerecht wird, bezweifeln manche. Es gibt bereits Ideen zu einer Neustrukturierung der Adressierung von Netzen und Host-Rechnern.

### 1.2.3 Client-Server-Technologie

Für die einzelnen Internet-Dienste wie World Wide Web, Gopher, E-Mail, FTP usw. muss auf einem Host-Rechner, der anderen Rechnern diese Dienste anbieten will, eine entsprechende **Server-Software** laufen. Ein Host-Rechner kann einen Internet-Dienst nur anbieten, wenn eine entsprechende Server-Software auf dem Rechner aktiv ist, wenn der Rechner »online« ist, und wenn keine schützende Software (Firewall) den Zugriff von außen verhindert bzw. einschränkt.

Server sind Programme, die permanent darauf warten, dass eine Anfrage eintrifft, die ihren Dienst betreffen. So wartet etwa ein Web-Server darauf, dass Anfragen eintreffen, die Web-Seiten auf dem Server-Rechner abrufen wollen.

**Clients** sind dagegen Softwareprogramme, die typischerweise Daten von Servern anfordern. Ihr Web-Browser ist beispielsweise ein Client. Wenn Sie etwa auf einen Verweis klicken, der zu einer HTTP-Adresse führt, startet der Browser, also der Client, eine Anfrage an den entsprechenden Server auf dem entfernten Host-Rechner. Der Server wertet die Anfrage aus und sendet die gewünschten Daten. Um die Kommunikation zwischen Clients und Servern zu regeln, gibt es entsprechende Protokolle. Client-Server-Kommunikation im Web etwa regelt das HTTP-Protokoll. Ein solches Protokoll läuft oberhalb des TCP/IP-Protokolls ab.





1.1: Das Internet setzt sich aus Host- und Client-Rechnern zusammen.

Dass ein Client Daten anfordert und ein Server die Daten sendet, ist der Normalfall. Es gibt jedoch auch Ausnahmen. So kann ein Client nicht nur Daten anfordern, sondern auch Daten an einen Server schicken zum Beispiel wenn Sie per FTP eine Datei auf den Server-Rechner hochladen, wenn Sie eine E-Mail versenden oder im Web ein Formular ausfüllen und abschicken. In diesen Fällen spricht man auch von **Client-Push** (»Client drängt dem Server Daten auf«).

Ein anderer Ausnahmefall liegt vor, wenn der Server zuerst aktiv wird und dem Client etwas ohne dessen Anforderung zuschickt. Das nennt man **Server-Push** (»Server drängt dem Client Daten auf«). Einige Technologien wollten diesen Ausnahmefall vor einigen Jahren zu einer Regel erheben mit den so genannten **Push-Technologien**. Diese Technologien sollten ermöglichen, dass ein Client regelmäßig Daten empfangen kann, ohne diese eigens anzufordern. Das ermöglicht Broadcasting-Dienste wie aktuelle Nachrichten. Netscape und Microsoft Internet Explorer (beide ab Version 4.0) bauten entsprechende Schnittstellen, um solche Dienste in Anspruch zu nehmen. Man kann jedoch von Glück sagen, dass sie sich nicht durchgesetzt haben. Das Internet ist eben nicht Fernsehen, sondern ein Medium, das von der Aktivität der Anwender lebt, nicht von deren Konsumbeflissenheit.

## 1.2.4 DNS – Domain Name Service

Computer können mit Zahlen besser umgehen, Menschen in der Regel besser mit Namen. Deshalb hat man ein System erdacht, das die numerischen IP-Adressen für die Endanwender in anschauliche Namensadressen übersetzt.

Dieses System ist ähnlich wie das der IP-Adressen hierarchisch aufgebaut. Eine Namensadresse (Domain-Name) in diesem System gehört zu einer **Top-Level-Domain**. Die einzelnen Teile solcher Namensadressen sind wie bei IP-Adressen durch Punkte voneinander getrennt. Namensadressen (Domains) sind beispielsweise *yahoo.com*, *mozilla.org* oder *teamone.de*.

Top-Level-Domains stehen in so einem Domain-Namen an letzter Stelle. Es handelt sich um einigermaßen sprechende Abkürzungen. Die Abkürzungen, die solche Top-Level-Domains bezeichnen, sind entweder Landeskennungen oder Typenkennungen. Beispiele sind:

- `de` = Deutschland
- `at` = Österreich
- `ch` = Schweiz
- `it` = Italien
- `my` = Malaysia
- `com` = Kommerziell orientierter Namensinhaber
- `org` = Organisation
- `net` = Allgemeines Netz
- `edu` = Amerikanische Hochschulen
- `gov` = Amerikanische Behörden
- `mil` = Amerikanische Militäreinrichtungen

Derzeit (Redaktionszeitpunkt dieses Buches) entstehen neue Top-Level-Domains. Geplant sind:

- `biz` = Unternehmen
- `pro` = Berufsgruppen mit Werbeverbot (Anwälte, Steuerberater, Ärzte usw.)
- `name` = Privatpersonen
- `info` = Informationsdienste aller Art
- `museum` = Museen
- `aero` = Flugunternehmen, Flughäfen, Reiseveranstalter usw.
- `coop` = Genossenschaften, Verbände, Organisationen

Zum Redaktionszeitpunkt dieses Buches sind `info` und `biz` bereits verfügbar, `name` soll als Nächstes folgen.

Jede Top-Level-Domain stellt einen Verwaltungsbereich dar, für den es eine »Verwaltungsbehörde« gibt, die für die Namensvergabe von Domains innerhalb ihres Verwaltungsbereichs zuständig ist. Wenn Sie beispielsweise einen Domain-Namen wie `MeineFirma.de` beantragen wollen, muss der Antrag an das **DENIC** (Deutsches Network Information Center) gestellt werden. Kommerzielle Provider erledigen das für Sie, wenn Sie dort einen entsprechenden Service in Anspruch nehmen. Ihren Wunschnamen erhalten Sie aber nur, wenn die Namensadresse noch nicht anderweitig vergeben wurde. Schlaufüchse sind daher auf die Idee gekommen, Tausende Domain-Namen für sich zu reservieren, um sie dann an Interessenten weiterzuverkaufen. Mittlerweile gibt es um die Domain-Namen leider schon eine ganze Latte von Schauergeschichten und Rechtsstreitigkeiten. Wenn etwa zwei zufällig gleichnamige Firmen, die sonst nichts miteinander zu tun haben, den gleichen Domain-Namen reservieren lassen wollen, kann nur eine der Firmen den Zuschlag erhalten. Beliebt geworden sind angesichts der Namensknappheit mittlerweile auch längere Namensadressen wie `heute-geh-ich-ins-kino.de`.

Inhaber von Domain-Namen können nochmals Sub-Level-Domains vergeben. So gibt es beispielsweise eine Domain namens `seite.net`. Die Betreiber dieser Domain haben nochmals Sub-

Domains vergeben, wodurch Domain-Adressen wie `java.seite.net` oder `javascript.seite.net` entstanden.

### 1.2.5 Routing und Gateways

Im Internet als dem Netz der Netze ist es zunächst nur innerhalb des eigenen Sub-Netzes möglich, Daten direkt von einer IP-Adresse zu einer anderen zu schicken. In allen anderen Fällen, wenn die Daten an eine andere Netzwerknummer geschickt werden sollen, treten Rechner auf den Plan, die den Verkehr zwischen den Netzen regeln. Solche Rechner werden als **Gateways** bezeichnet. Diese Rechner leiten Daten von Host-Rechnern aus dem eigenen Sub-Netz an Gateways in anderen Sub-Netzen weiter und ankommende Daten von Gateways anderer Sub-Netze an die darin adressierten Host-Rechner im eigenen Sub-Netz. Ohne Gateways gäbe es gar kein Internet.

Das Weiterleiten der Daten zwischen Sub-Netzen wird als **Routing** bezeichnet. Die Beschreibung der möglichen Routen vom eigenen Netzwerk zu anderen Netzwerken sind in **Routing-Tabellen** auf den Gateway-Rechnern festgehalten.

Zu den Aufgaben eines Gateways gehört auch, eine Alternativ-Route zu finden, wenn die übliche Route nicht funktioniert, weil etwa bei der entsprechenden Leitung eine Störung oder ein Datenstau aufgetreten ist. Gateways senden sich ständig Testpakete zu, um das Funktionieren der Verbindung zu testen und für Datentransfers »verkehrsarme« Wege zu finden.

Wenn also im Internet ein Datentransfer stattfindet, ist keinesfalls von vorneherein klar, welchen Weg die Daten nehmen. Sogar einzelne Pakete einer einzigen Sendung können völlig unterschiedliche Wege nehmen. Wenn Sie beispielsweise von Deutschland aus eine Web-Seite aufrufen, die auf einem Rechner in den USA liegt, kann es sein, dass die Hälfte der Seite über den Atlantik kommt und die andere über den Pazifik, bevor Ihr Web-Browser sie anzeigen kann. Weder Sie noch Ihr Browser bekommen davon etwas mit.

### 1.2.6 Selbstorganisation im Internet

In Anbetracht der Teilnehmerzahl im Internet ist der Verwaltungsaufwand im Netz vergleichsweise klein. Die meisten Endanwender wissen nicht einmal, dass es solche Verwaltungsstellen überhaupt gibt. Eine gesetzgeberische Institution, wie es sie etwa innerhalb des Verfassungsereichs eines Staates gibt, gibt es im Internet als weltweitem Verbund nicht. Der Versuch, die ICANN-Behörde als oberste Vergabestelle für Domains und IPs zu etablieren, hat viel Staub aufgewirbelt, aber die Befürchtungen, dass da eine zentralistische Internet-Diktatur entsteht, haben sich bislang nicht bestätigt. Viele Bereiche im Internet beruhen faktisch auf Selbstorganisation. Bei Diensten wie E-Mail gilt beispielsweise das stille Abkommen, dass jeder beteiligte Gateway alle Mails weiterleitet, auch wenn weder Sender noch Empfänger dem eigenen Sub-Netz angehören. Die Kosten trägt zwar der Netzbetreiber, aber da alle ein Interesse am weltweiten Funktionieren des Netzes haben, ist jeder bereit, die anfallenden Mehrkosten zu tragen.

Die Funktionsfähigkeit des Internet basiert also auf der Bereitschaft der Beteiligten, keine Pfennigfuchserie zu betreiben. Großzügigkeit hat das Internet geschaffen, und Kleingeisterei ist der größte Feind der Internet-Idee.

Das Usenet, also der größte Teil der Newsgroups, organisiert sich sogar vollständig selbst, weshalb leidenschaftliche Anhänger dieses System gerne als real existierendes Beispiel für Herrschaftsfreiheit anführen. Die »Verwaltung« findet im Usenet in speziellen Newsgroups statt (solchen, die mit `news .` beginnen). Dort können beispielsweise Vorschläge für neue Gruppen eingebracht werden, und in Abstimmungen wird darüber entschieden, ob eine Gruppe eingerichtet oder abgeschafft wird.

Offizielle Anlaufstellen gibt es für die Vergabe von Netzwerkadressen und für Namensadressen (DNS). Für die Vergabe von IP-Adressen innerhalb eines Netzwerks ist der jeweilige Netzbetreiber verantwortlich. Dazu kommen Organisationen, die sich um Standards innerhalb des Internet kümmern.

Die Kosten für die Datenübertragungen im Internet tragen die Betreiber der Sub-Netze. Diese Kosten pflanzen sich nach unten fort zu Providern innerhalb der Sub-Netze bis hin zu Endanwendern, die über Provider Zugang zum Internet haben oder Internet-Services wie eigene Webseiten nutzen. Die folgende Liste enthält einige Verweise zu den wichtigsten internationalen und nationalen Organisationen im Internet:

- Deutsches Network Information Center (DE-NIC, [www.denic.de/](http://www.denic.de/)): Vergabestelle für Domain-Namen unterhalb der Top-Level-Domain `.de`.
- International Network Information Center (InterNIC, [www.internic.net/](http://www.internic.net/)): Kollaboratives Projekt zwischen AT&T, General Atomics und Network Solutions, Inc. AT&T. Internationale Vergabestelle für Domain-Namen.
- The Internet Corporation for Assigned Names and Numbers (ICANN, [www.icann.org/](http://www.icann.org/)): Organisation für die Schaffung von neuen Top-Level-Domains und für die Vergabe von Subnetzen und IP-Adressräumen.
- Internet Architecture Board (IAB, [www.iab.org/](http://www.iab.org/)): Organisation zur Dokumentation der Netzstruktur und der grundsätzlichen Abläufe im Internet.
- Internet Assigned Numbers Authority (IANA, [www.iana.org/](http://www.iana.org/)): Zentrale Koordinationsstelle für Internet-Protokolle.
- Internet Engineering Task Force (IETF, [www.ietf.org/](http://www.ietf.org/)): Internationale Gemeinschaft von kommerziellen und nichtkommerziellen Aktiven im Internet mit dem Ziel, technische Standards im Internet vorzuschlagen.
- Internet Society ([www.isoc.org/](http://www.isoc.org/)): Internationale Organisation für die Kooperation und Koordination von Technologien und Anwendungen im Internet.
- W3-Konsortium (W3C, [www.w3.org/](http://www.w3.org/)): Organisation, die speziell die Weiterentwicklung technischer Standards des World Wide Web koordiniert, etwa HTML oder das HTTP-Protokoll.

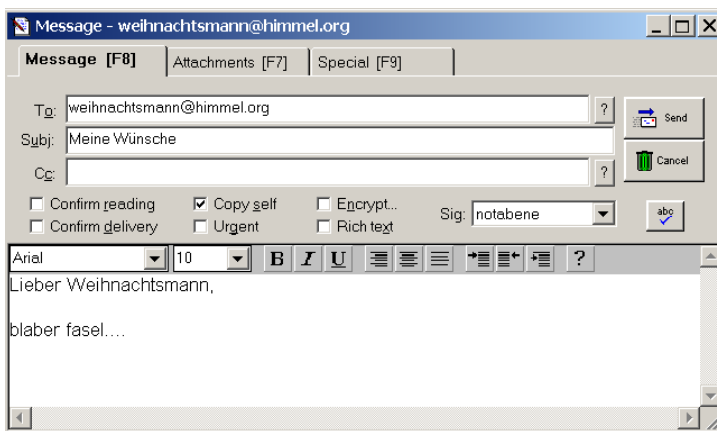
Vor allem jene Organisationen, die sich um die technische Weiterentwicklung im Internet kümmern, werden zunehmend von großen Softwarefirmen wie Microsoft, Netscape oder Sun bestürmt, da diese Firmen ein Interesse daran haben, ihren Softwareprodukten und hauseigenen Standards bei Server-Technologien, Programmiersprachen usw. zum Status weltweiter Internet-Standards zu verhelfen. Ob und wie weit es gelingt, im Internet angesichts des entstehenden Milliardenmarkts neue, firmenunabhängige Standards durchzusetzen, die so erfolgreich werden wie HTML oder HTTP, muss die Zukunft zeigen. Derzeit sieht es gar nicht schlecht aus. Die

meisten Computerkonzerne haben begriffen, dass firmenunabhängige Standards ihnen selber letztlich auch mehr bringen. Und der Druck aufgeklärter Anwender, die keine Lust mehr haben, wegen allem und jedem eine neue, proprietäre Software kaufen zu müssen, die nach ein paar Jahren wieder »out« ist und vom Markt verschwindet, wächst ebenfalls.

## 1.3 Dienste im Internet

### 1.3.1 E-Mail

E-Mail (elektronische Post) ist wohl der am meisten genutzte Internet-Dienst. E-Mail erlaubt die persönliche Übermittlung von Nachrichten und Dateien von einem Sender an einen Empfänger. Wer an diesem Dienst teilnehmen will, braucht eine eigene E-Mail-Adresse. Solche Adressen sind an dem berühmten @ (sprich: englisch »at«, also »bei« – das Zeichen wird im deutschen Sprachraum auch als »Klammeraffe« bezeichnet) in der Mitte erkennbar. Provider, über die Sie Zugang zum Internet haben oder bei denen Sie Ihr Web-Angebot hosten, richten Ihnen normalerweise E-Mail-Adressen auf Wunsch ein.



1.2: E-Mail ist der meistgenutzte Dienst im Internet.

Vor allem im Business-Bereich verdrängt E-Mail nach und nach die herkömmliche Briefpost (von E-Mail-Anhängern liebevoll oder verächtlich als »Schneckenpost« oder »snail mail« bezeichnet). Auch das Fax ist eine durch E-Mail gefährdete Gattung. Die Vorteile liegen auf der Hand:

- E-Mails brauchen meist nur wenige Minuten vom Sender zum Empfänger, egal, ob einige wenige oder mehrere tausend Kilometer zwischen ihnen liegen.
- E-Mails sind preiswert. Im Vergleich zu Briefen fallen für einfache E-Mails nur Pfennigbeträge an.
- Gegenüber einem Telefonat bieten sie den Vorteil, dass Vereinbarungen oder dergleichen darin schriftlich festgehalten werden.

- Moderne E-Mail-Programme können E-Mails in einer durchsuchbaren Datenbank speichern, sodass Sie vor längerer Zeit gesendete oder empfangene Mails leicht wiederfinden.
- Der Mime-Standard, der sich bei E-Mails allmählich durchsetzt, erlaubt das bequeme Anhängen beliebiger Computerdateien an eine Mail, sodass E-Mail auch für den individuellen Austausch von Dateien immer mehr Bedeutung gewinnt.
- E-Mail zwingt Sender und Empfänger nicht, gleichzeitig online zu sein. Sie holen sich Ihre E-Mails ab, wenn Sie gerade Zeit haben.
- Neben der Möglichkeit, mit bestimmten anderen Internet-Teilnehmern Nachrichten und Daten auszutauschen, können Sie mit einer eigenen E-Mail-Adresse auch an **Mailing-Listen** teilnehmen, die Sie interessieren.

Das heutige E-Mail-System hat aber auch noch mit einigen Problemen zu kämpfen. Eine normale E-Mail ist auf dem Weg vom Sender zum Empfänger etwa so geheim wie eine Ansichtskarte. Für vertrauliche Mitteilungen oder sensible Daten ist sie ungeeignet. Mittlerweile gibt es Verschlüsselungsverfahren wie **PGP (Pretty Good Privacy)**, die das individuelle Kodieren und Dekodieren von E-Mails und angehängten Dateien erlauben. Voraussetzung ist dazu jedoch, dass sowohl Sender als auch Empfänger über eine entsprechende Zusatzsoftware verfügen und zuvor ihre öffentlichen Kodierschlüssel austauschen.

Ein weiterer Nachteil von E-Mails ist auch noch, dass sie nicht als »dokumentenecht« gelten. Zwar wächst allmählich ihre juristische Akzeptanz, doch unterschriftspflichtige Dokumente sind durch E-Mail bislang nicht ersetzbar. Deshalb können Sie z.B. noch nicht »einfach so« per E-Mail einen Wohnsitzwechsel mitteilen oder Ihre Steuererklärung ans Finanzamt schicken.

### 1.3.2 Telnet und SSH

Telnet ist dazu gedacht, einen fernen Rechner im Internet so zu bedienen, als säße man direkt davor. Telnet ist damit eine einfache Lösung für Teleworker. Und diese Lösung ist schon wesentlich älter als der Begriff »Teleworking«.

Telnet ist vor allem für Unix-Systeme gedacht. Es erlaubt das betriebssystemeigene **login** (Anmelden) eines Benutzers an einem ans Internet angeschlossenen Host-Rechner in Form eines **rlogin** (remote login). Das Anmelden ist nur möglich, wenn Sie Username und Passwort kennen, d.h. auf dem angewählten Host-Rechner als Benutzer eingetragen sind. Nach dem Einwählen erhalten Sie eine Unix-Shell (Eingabeaufforderung) und können auf dem entfernten Rechner Betriebssystembefehle eingeben, Programme starten usw.

PC-Anwender, die nicht direkt mit der Verwaltung von Server-Rechnern im Internet zu tun haben, werden mit Telnet kaum in Berührung kommen. Es gibt jedoch auch für PC-Benutzer von Systemen wie MS Windows oder Macintosh **Telnet-Clients**. Diese Programme erlauben es, vom eigenen PC aus auf einem entfernten Host-Rechner zu arbeiten. Ohne Kenntnis der Befehle des Host-Rechner-Betriebssystems ist ein solches Programm allerdings zwecklos.

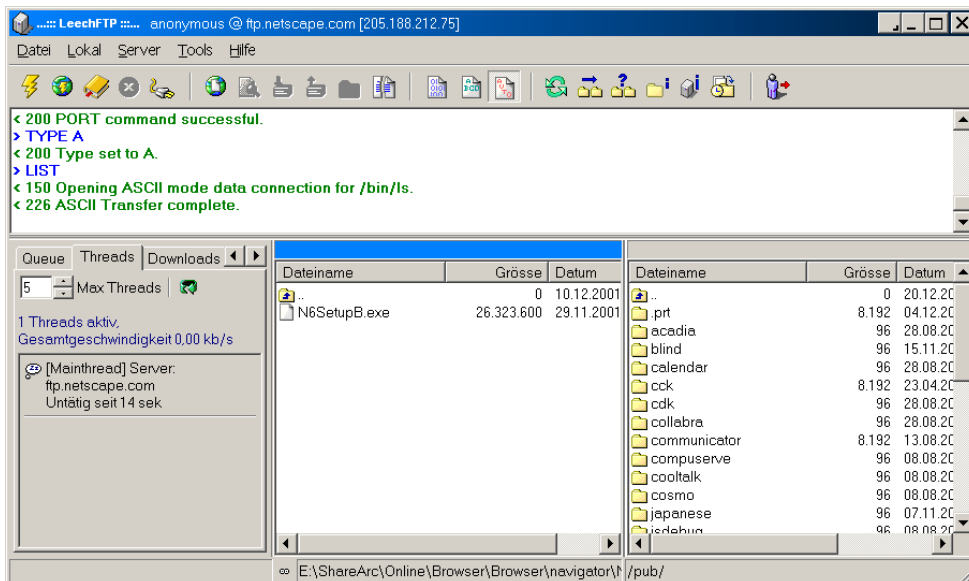
**SSH (Secure Shell)** ist eine Variante von Telnet – die bessere Variante, wie allmählich erkannt wird. Der Unterschied zu Telnet ist, dass bei SSH die Daten beim Arbeiten verschlüsselt übertragen werden. Das ist durchaus wichtig, da gerade beim Arbeiten auf einem entfernten Rechner häufig sensible Daten editiert werden, wie Konfigurationsdateien oder Passwörter. All diese Daten werden – auch wenn es einem beim Arbeiten kaum bewusst ist – über viele Rechner im

Internet bis zum Zielrechner übertragen und können auf den Zwischenstationen mit etwas bösem Willen abgefangen werden. Manche modernere Telnet-Clients bieten mittlerweile auch das SSH-Protokoll an.

### 1.3.3 File Transfer Protocol (FTP)

FTP ist ein Internet-Dienst, der speziell dazu dient, sich auf einem bestimmten Server-Rechner im Internet einzuwählen und von dort Dateien auf den eigenen Rechner zu übertragen (**Download**) oder eigene Dateien an den Server-Rechner zu übertragen (**Upload**). Ferner bietet das FTP-Protokoll Befehle an, um auf dem entfernten Rechner Operationen durchzuführen wie Verzeichnisinhalte anzeigen, Verzeichnisse wechseln, Verzeichnisse anlegen oder Dateien löschen.

Beim Einwählen an einem FTP-Server gilt es, zwei Arten zu unterscheiden: Es gibt »normales« FTP und **anonymes FTP**. Beim normalen FTP haben Sie nur Zugriff auf den Server, wenn Sie beim Einwählen einen Username und ein Passwort angeben. Mit diesen Zugangsdaten muss der FTP-Server-Verwalter Sie am Server als berechtigten Anwender eingetragen haben. Bei anonymem FTP handelt es sich um öffentlich zugängliche Bereiche auf Server-Rechnern. Dazu brauchen Sie keine Zugangsberechtigung. Sie wählen sich einfach mit dem Username `anonymous` ein, und als Passwort geben Sie Ihre E-Mail-Adresse an. Es gibt etliche öffentliche FTP-Server im Internet, die umfangreiche Dateibestände zum Download anbieten. Dort finden Sie Software, Hilfetexte, Grafiksammlungen usw.



1.3: Mit einem FTP-Programm übertragen Sie schnell Dateien über das Internet.

Für öffentliche, anonyme FTP-Server gibt es Regeln. Da Sie beim Besuch solcher Server nur Gast sind, sollten Sie sich auch entsprechend verhalten. Bleiben Sie nicht länger als nötig. Laden Sie sich nicht unnötig viele Dateien herunter. Lesen Sie zuerst die Readme-Dateien, die es in fast jedem Verzeichnis gibt, um den Inhalt des Verzeichnisses zu erläutern.

Wenn Sie lediglich hin und wieder öffentlich zugängliche FTP-Server besuchen, um von dort Dateien downzuloaden, können Sie das auch mit modernen WWW-Browsern wie Netscape oder dem MS Internet Explorer tun. Solche Browser zeigen die Dateilisten auf FTP-Servern als anklickbare Verweise an. Je nach Dateityp können Sie Dateien dann downloaden oder direkt im Browser-Fenster anzeigen.

Für anspruchsvollere Arbeiten, vor allem, wenn Sie die Dateien Ihres eigenen Web-Projekts verwalten wollen, brauchen Sie ein FTP-Programm. Solche Programme gibt es für alle Betriebssysteme, und einige Betriebssysteme, etwa alle Unix-Derivate oder OS/2, haben bereits einen eingebauten FTP-Client.

### 1.3.4 Gopher

Gopher gilt als der Vorläufer des World Wide Web. Der Name kommt von »go for« und drückt damit aus, was der wichtigste Zweck dieses Dienstes ist: nämlich große Informationsbestände leichter durchsuchbar zu machen. Gopher ist eine menübasierte Bedienoberfläche zum Auffinden von Information, aber auch zum Nutzen anderer Internet-Dienste wie FTP oder Telnet. Einem Eintrag in einem Gopher-Menü ist nicht anzusehen, wo sich die damit verbundenen Dateien oder Programme befinden. Anwender mit text- und tastaturorientierten Rechnern können aus den Menüs, die das Gopher-System am Bildschirm anzeigt, mit Buchstaben- oder Zifferntasten Einträge auswählen. Bei grafischen Benutzeroberflächen sind die Menüs mit der Maus anklickbar. In dieser Hinsicht gleicht Gopher dem World Wide Web, denn auch bei diesem Dienst müssen Sie keine Insider-Befehle kennen, um im Informationsbestand zu navigieren.

Der Nachteil von Gopher gegenüber dem WWW ist, dass es keine Standards wie HTML gibt. Es gibt lediglich die Gopher-Menüs und die damit verknüpften Dateiaufrufe oder Befehle. Ein guter Gopher-Client kann zwar diverse Dateiformate anzeigen, doch es gibt kein Mittel, um Bildschirmseiten zu gestalten und dadurch eigenständige Präsentationen von Information zu schaffen.

Moderne Web-Browser sind auch Gopher-fähig. Das bedeutet, dass Sie mit einem solchen Browser problemlos Gopher-Adressen aufrufen können. Die Gopher-Menüs erscheinen im Browser wie Listen mit Verweisen in HTML.

### 1.3.5 Chat (IRC)

Wer sich einsam fühlt oder einfach »in« sein will, geht im Internet **chatten** (ratschen, quatschen). Am Bildschirm erscheint dann ein zweigeteiltes Fenster. Im einen Teil erscheinen wie von Geisterhand geschrieben allerlei ganze und halbe Sätze, Kommentare und solche Dinge wie Emoticons. Das sind Beiträge von Teilnehmern, die gerade an der gleichen Stelle online sind. Im anderen Fenster können Sie selbst etwas eintippen. Auf diese Weise können Sie an der Unterhaltung teilnehmen. Es gibt auch grafische Oberflächen, bei denen sich jeder Chat-Teilnehmer eine Figur aussucht, die dann als Teilnehmer in einer Szenerie erscheint.

Kaum jemand erscheint dort mit seinem wahren Namen, und nicht wenige geben sich als etwas ganz anderes aus, als sie wirklich sind. Männer spielen Frau, um herauszufinden, wie es ist, als Frau von einem Mann angemacht zu werden. Erwachsene spielen Jugendliche, um sich mal wieder richtig pubertär benehmen zu dürfen. Hin und wieder kommen interessante Gespräche



zustande, aber oft haben die Chat-Beiträge auch das Niveau von Kindergestammel. Über Chats entstehen aber auch ernst zu nehmende Kontakte, Freundschaften und Beziehungen.

IRC ist einer der Internet-Dienste, die teuer werden können. Denn während der ganzen Teilnahmedauer muss man online sein, und viele merken beim Quatschen nicht, wie die Zeit vergeht (das ist nicht anders als im »real life«). Es gibt nur wenige, die eine ganze Nacht lang im Web surfen, aber viele, die eine ganze Nacht lang chatten. Viele der hochgradig Internet-Süchtigen treiben sich vor allem in den Chat-Bereichen herum.

### 1.3.6 Newsgroups (News)

Eine Newsgroup ist einem Schwarzen Brett vergleichbar, wo Sie Nachrichten posten können, die alle Besucher lesen können. Jede Newsgroup behandelt einen Themenbereich. Insgesamt sind mehrere zigtausend Newsgroups registriert. Es gibt praktisch nichts, zu dem es nicht eine Newsgroup gibt. Egal, ob Sie sich für die Konfiguration Ihres PCs interessieren, für Origami oder für vermisste Kinder – für alles finden sich eine oder mehrere Newsgroups. Täglich werden zigtausend Nachrichten in Newsgroups gepostet. Es werden Fragen gestellt und Antworten gegeben, es wird debattiert und geflachst. Die Newsgroups gelten allgemein als der verrückteste Teil des Internet. Manchen Leuten sind sie aber auch ein Dorn im Auge, denn es gibt auch etliche Newsgroups mit pornographischen und extremistischen Inhalten.

Das System der Newsgroups ist auf verschiedene Netze verteilt. Das größte und bekannteste ist das **Usenet**. Hier finden Sie Newsgroups mit Adressen wie `alt.music.pinkfloyd` oder `de.soc.weltanschauung`. Die wichtigsten Abkürzungen in solchen Newsgroup-Adressen sind:

- `alt` = alternativ, bunt, verrückt, abgefahren
- `biz` = Kommerzielles, jedoch keine Werbung
- `comp` = Computer
- `de` = deutschsprachig
- `misc` = Sonstiges
- `news` = Newsgroups zum Thema Newsgroups
- `rec` = Freizeit, Hobby und Kunst
- `sci` = Wissenschaften
- `soc` = Soziales, Kultur, Politik
- `talk` = Klatsch und Tratsch

Um Newsgroups lesen und daran teilnehmen zu können, brauchen Sie ein **Newsreader**-Programm. Web-Browser wie Netscape (ab V3.0) oder der MS Internet Explorer (ab V4.0) haben einen eingebauten News-Client, sodass Sie kein separates Programm mehr benötigen. Um Newsgroups empfangen zu können, müssen Sie in den Einstellungen des verwendeten Programms einen **News-Server** angeben. Fragen Sie den Provider, über den Sie Ihren Internet-Zugang haben, ob er auch einen Newsgroup-Server anbietet und mit welchen Daten Sie sich dort anmelden können.

Es empfiehlt sich, zunächst einmal alle verfügbaren Newsgroups einzulesen. Das kann eine Weile dauern, aber anschließend haben Sie eine sortierte Liste aller Gruppen und können sich einen Überblick verschaffen. Newsgroups, die Sie interessieren, können Sie **abonnieren**.

Nachrichten in Newsgroups sind hierarchisch organisiert. Irgendjemand postet eine Nachricht mit einem neuen **Subject** (Thema). Ein anderer antwortet auf die Nachricht und bezieht sich dadurch auf das gleiche Thema. Ein dritter antwortet ebenfalls, und ein vierter antwortet dem Dritten. Auf diese Weise entsteht eine Baumstruktur von Nachrichten zu einem einmal begonnenen Thema. Diese Baumstruktur wird auch als **Thread** bezeichnet. Sie können selbst einen neuen Thread beginnen oder auf eine Nachricht antworten und sich dadurch in den betreffenden Thread einschalten.

### 1.3.7 World Wide Web (WWW)

Das World Wide Web (WWW) ist der jüngste Dienst innerhalb des Internet. Das Web zeichnet sich dadurch aus, dass es auch ungeübteren Anwendern erlaubt, sich im Informationsangebot zu bewegen. Wer etwa mit einem FTP-Programm einen FTP-Server aufruft, muss sich in komplexen, unbekannten Verzeichnisstrukturen zurechtfinden und sich an Dateinamen orientieren. Interessante Dateien kann er auf seinen Rechner downloaden, um sie später zu öffnen. Im WWW dagegen erscheinen Informationen gleich beim Aufruf am Bildschirm. Wenn Sie mit einem Browser im Web unterwegs sind, brauchen Sie sich nicht um Dateinamen oder um komplizierte Eingabebefehle zu kümmern. Das bequeme Navigieren mit Hilfe einfacher Mausklicks kann ein Hochgefühl erzeugen, weshalb sich auch die Redewendung vom »Surfen im Netz« eingebürgert hat.

## 1.4 Entstehung des World Wide Web

### 1.4.1 Tim Berners-Lee und die Anfänge

Der gebürtige Brite **Tim Berners-Lee**, in den 80er Jahren und bis in die 90er Jahre Informatiker am Genfer Hochenergieforschungszentrum CERN, beschreibt in seinem Buch »Weaving the Web«, wie er dort am CERN in den 80er Jahren ein selbst geschriebenes Programm als Produktivitäts-Tool nutzte. Das Programm, von Berners-Lee in Pascal geschrieben, lief auf einem Computersystem mit dem Betriebssystem »Norsk Data SYNTRAN III«. Die 8-Zoll-Diskette, auf der die Quellen gespeichert waren, gingen irgendwann verloren.



1.4: Tim Berners-Lee, der Vater des World Wide Web

Mit heutiger Software hatte das Tool sicher nicht viel zu tun. Grafische Oberflächen gab es nicht, kommandozeilenorientierte Systeme dominierten, und grüne Schrift auf schwarzem Grund beherrschte die Bildschirme. Auf einem solchen System also hatte Tim Berners-Lee sein Pascal-Programm zum Laufen gebracht. Er nannte das Programm **Enquire**, was auf Deutsch so viel bedeutet wie »sich erkundigen«.

»Enquire« war ein Hypertext-Programm. Man konnte Textdateien editieren, die – vermutlich durch irgendwelche Steuerzeichenfolgen markiert – in »nodes« (Knoten) unterteilt waren. Ein Knoten konnte alles sein, was so an Daten anfiel – Adressen, Gesprächsnotizen, spontane Ideen, Erlebnisse, Arbeitsergebnisse. Zu jedem Knoten gab es eine zugehörige Liste mit Links zu anderen Knoten. Man konnte auf jegliche Art von Querbeziehung linken, die man kannte oder fand. Links auf Ziele innerhalb einer Datei wurden vom Enquire-Programm automatisch bidirektional dargestellt – das heißt, auch wenn der Link nur von A nach B gesetzt war, fand man bei B in der Liste den Rückverweis auf A.

Ende 1988 entschloss sich Berners-Lee, aus der Weiterentwicklung von »Enquire« ein computerübergreifendes System zu entwickeln. Für CERN sollte er zu diesem Zweck einen entsprechenden Projektvorschlag einreichen und darin sein Vorhaben detailliert beschreiben. Das war die Zeit, als Berners-Lee mit verschiedenen Fachleuten Kontakt aufnahm, um technische Wege der Realisierung seiner Ideen zu finden. Ben Segal, ein Kollege von Berners-Lee, der sich mit den Grundlagen des Internet auskannte, überzeugte den Hypertext-Entwickler von den einzigartigen Möglichkeiten des Netzes der Netze. 1989 reichte er seinen Vorschlag bei CERN ein. Es vergingen jedoch viele Monate, in denen Berners-Lee keine Reaktion auf den Vorschlag erhielt. Robert Cailliau, ein alter Bekannter, ließ sich von den Ideen des Briten begeistern und setzte sich fortan für die Forcierung des Projekts am CERN ein. In dieser Zeit – es war das Jahr 1990 – erhielt das Projekt auch seinen endgültigen Namen: **World Wide Web**. Im Herbst des Jahres 1990 schrieb Berners-Lee eigenhändig die ersten Versionen der drei Säulen seines Konzepts:

- Die Spezifikation für die Kommunikation zwischen Web-Clients und Web-Servern – das so genannte **HTTP-Protokoll** (*HTTP = Hypertext Transfer Protocol*).
- Die Spezifikation für die Adressierung beliebiger Dateien und Datenquellen im Web und im übrigen Internet – das Schema der so genannten **URIs** (*URI = Universal Resource Identifier, universeller Quellenbezeichner*).
- Die Spezifikation einer Auszeichnungssprache für Web-Dokumente, der Berners-Lee den Namen **HTML** gab (*HTML = Hypertext Markup Language, Hypertext Auszeichnungssprache*).

Berners-Lee schrieb auch die erste Web-Server-Software. Der Rechner, auf dem diese Software installiert wurde, war unter dem Namen `info.cern.ch` erreichbar. Dort stellte Berners-Lee an Weihnachten 1990 die ersten, in HTML geschriebenen Web-Seiten der Welt zur Verfügung. Seine Ideen wichen dabei ursprünglich durchaus von dem ab, was schließlich aus dem Web wurde. So setzte sich Berners-Lee immer dafür ein, Web-Seiten online editierbar zu machen, sodass Web-Seiten-Besucher Texte fortschreiben konnten, sofern der Anbieter entsprechende öffentliche Schreibrechte für die Dateien vergab. Doch die Web-Browser, die sich schließlich durchsetzten, waren reine Lese-Software.

Das Web entwickelte sich zunächst nicht unbedingt von allein. Berners-Lee und seine Projektmitstreiter waren unermüdlich bei der Arbeit, das Web bekannt zu machen und zu etablieren. Auf einer internationalen Hypertext-Konferenz im Jahre 1991 stellten sie das Projekt vor. Kontakte zu Programmierern für verschiedene Systeme entstanden, und nur so entstanden die ersten

Web-Browser. Denn erst die Verfügbarkeit solcher Browser ermöglichte es anderen Menschen mit Internet-Zugang, Web-Seiten aufzurufen. Nicola Pellow, eine junge Mitarbeiterin und Mathematikerin am CERN, schrieb den ersten einigermaßen benutzbaren textmodusorientierten Browser. 1992 entstanden auch die ersten Browser für grafische Benutzeroberflächen, mit den Namen *Erwise* und *Viola*. Gleichzeitig stieg die Zahl von Web-Servern, über die Web-Seiten angeboten wurden. Zunächst waren es vor allem wissenschaftliche Institutionen.

### 1.4.2 Marc Andreessen, Netscape und der Boom

Für programmierende Studenten wurde es zunehmend zu einer interessanten Herausforderung, Browser fürs Web zu schreiben. Besonders hervor tat sich dabei ein junger Mann namens **Marc Andreessen**. Er entwickelte einen Web-Browser für grafische Benutzeroberflächen namens **Mosaic**. Tim Berners-Lee, der Marc Andreessen auch selber kennen lernte, beschreibt ihn als jemanden, der nicht wie alle Browser-Entwickler vor ihm eine Software fürs Web entwickeln wollte, sondern als jemanden, der mit dem Anspruch auftrat, das Web überhaupt erst verfügbar zu machen – eben mit seinem Browser.



1.5: Marc Andreessen, der den ersten brauchbaren grafischen Web-Browser Mosaic programmierte.

Andreessen und seine angeheuerten Mitprogrammierer hatten mit den philosophischen Ideen von Berners-Lee nicht sehr viel am Hut. Sie dachten von Beginn an in kommerziellen Kategorien. Bei der Entwicklung des Mosaic-Browsers erfanden die Programmierer nach ihrem Belieben neue Features, die sie in den Web-Browser implementierten, ohne dass diese Features standardisiert waren. Aber auch wenn Berners-Lee durch die Zeilen seines Buches deutlich durchscheinen lässt, wie wenig er von Andreessen hielt – durch den Mosaic-Browser kam die Lawine Web erst richtig ins Rollen. Das Mosaic-Projekt war dem ehrgeizigen Marc Andreessen jedoch alsbald zu klein geworden. Er wollte mehr, er wollte einen Browser, der die Welt erobert. Andreessen stieg schließlich aus dem Mosaic-Projekt aus und wurde Mitbegründer einer neuen Firma für Web-Software: **Netscape**. Wer heute den »guten« Netscape-Browser gegen den »bösen« Microsoft-Browser verteidigt, weiß also meistens gar nicht, dass Netscape in den Anfangszeiten genauso unverblühte Markteroberungs- und Monopolgelüste hatte wie Microsoft.

Das Konzept von Netscape ging auch zunächst voll auf. Eine völlig neue Zunft entstand: die **Web-Designer**. Träumend saßen sie in den Jahren 1995 und 1996 vor ihren ersten Web-Seiten, die dank Netscape bunte Hintergrund- und Schriftfarben, Hintergrundtapeten, Tabellenlayouts, mehrgeteilte Bildschirmfenster (Frames) und Multimedia-Plug-Ins enthalten konnten. Zigtausende von Privatanwendern begannen, eigene Homepages zu erstellen. Nach und nach drängten alle Firmen, Organisationen, Regierungen und Behörden mit eigenen Angeboten ins Web – weltweit. Die Wachstumsraten glichen einer Explosion. Es war aber auch die Stunde jener Mar-

keting-Strategen, die am liebsten aus der ganzen Welt eine Plakatwand machen würden. Mit Feuereifer und Finanzen stürzten sie sich auf das neue, aufstrebende Medium, ohne den geringsten Schimmer einer Ahnung von dessen wahrem Wesen zu haben. Kein Wunder, dass viele Versuche, die Sahnetorte Web aufzurollen, kläglich scheiterten und schließlich wieder für Katerstimmung am Werbemarkt und an der Börse sorgten.

In den Jahren 1995 und 1996 erreichte der Netscape-Browser unter den Web-Benutzern zeitweise einen Marktanteil von 90%. Der Microsoft-Konzern schlief zunächst noch den Schlaf der Seligen und fraß sich an den Umsatzzahlen seiner Betriebssysteme MS DOS und MS Windows und der Textverarbeitung MS Word satt. Mitte der 90er Jahre, als der Online-Boom allmählich breitere Schichten der Bevölkerung der westlichen Länder erreichte, setzte man bei Microsoft zunächst auf einen eigenen, proprietären Online-Dienst namens MSN. Vom Internet und dem Web behauptete Bill Gates, das sei nichts für Microsoft. Als man jedoch sah, wie Netscape binnen weniger Monate die EDV-Landschaft veränderte, wie die Firma Netscape an der Börse einen bis dahin noch nie dagewesenen Traumstart hinlegte und die Implementierungen des Netscape-Browsers sich vermehrten wie ein Computer-Virus, da begann man plötzlich hastig zu reagieren. Mit einem gewaltigen Aufwand an Manpower wurde Versäumtes aufgeholt. Innerhalb von etwas mehr als einem Jahr erschienen die ersten vier Versionen des MS Internet Explorer, wobei Microsoft im Frühjahr 1997 mit der 4er-Version ein ähnlicher technologischer Durchbruch gelang wie Netscape mit den 1995 erschienenen Versionen 1.1 und 2.0 seines Browsers. Microsoft war alles andere als zimperlich, was den Kampf gegen Netscape betraf. So erklärte man den eigenen Browser einfach als Teil des Betriebssystems MS Windows und sorgte über diesen vorhandenen Vertriebskanal für die gewaltige Verbreitung des Internet Explorer. Nun ist dagegen eigentlich nichts einzuwenden, denn schließlich gehört Internet-Client-Software seit jeher zum Umfang von Betriebssystemen, gerade auch in der Unix-Welt. Die Absichten von Microsoft waren jedoch nur allzuleicht durchschaubar und sorgten in der Internet-Gemeinde für eine Welle des Hasses gegen den Konzern aus Redmond.

### 1.4.3 Die Arbeit des W3-Konsortiums

Der Browser-Krieg tobt bis heute – zumindest überall dort, wo diskutiert wird. In den Logdateien der Web-Server ist er jedoch längst entschieden. Microsoft steuert heute im Browser-Markt allmählich eine ähnliche 80-90%-Machtstellung an wie im Bereich der Betriebssysteme. Doch es gibt auch gute Nachrichten: Das Bewusstsein für die technischen Grundlagen des Web ist gewachsen, und die Ideen von Tim Berners-Lee werden mittlerweile durch ein vielköpfiges Konsortium vertreten: das **W3-Konsortium (W3C)**. Dieses Konsortium ist trotz der millionenfach verbreiteten Browser der mächtigste Faktor für die Weiterentwicklung des Web geworden. Nicht zuletzt deshalb, weil es nicht gegen die Marktinteressen der Softwarefirmen gerichtet ist, sondern seine Mitglieder zu großen Teilen aus eben jenen Softwarefirmen rekrutiert.

Ende 1994 traf ein erstes Beratungskomitee zusammen, um die Grundlagen für eine Institution zu schaffen, die sich um die technischen Grundlagen und Standards im Web kümmern sollte. Das CERN war mit dieser Aufgabe, die nicht seinem eigentlichen Beschäftigungsgegenstand entsprach, einfach überfordert.

Im Sommer 1995 traten namhafte Firmen dem W3-Konsortium bei. Gleichzeitig eröffnete die europäische Präsenz des Konsortiums ihre Pforten. Mitglieder des W3-Konsortiums sind Firmen oder Organisationen, keine Einzelpersonen. Sie unterschreiben einen 3-Jahres-Vertrag und zah-

len Mitgliederbeiträge, aus denen sich das Gremium finanziert. Als Gegenleistung erhalten sie Zugang zu nichtöffentlichen Informationen und können an der Entwicklung der vom W3C betreuten Standards wie HTML, CSS, XML usw. mitwirken.

Die Arbeit des W3-Konsortiums unterteilt sich in so genannte **Aktivitäten** (*Activities*). Es gibt mehrere Dutzend solcher Aktivitäten. So gibt es Aktivitäten für HTML, XML, CSS usw. Für jede der Aktivitäten gibt es **Arbeitsgruppen** (*Working Groups*) und **Interessensgruppen** (*Interest Groups*). Während die Interessensgruppen eher Einfluss nehmenden Charakter haben, befassen sich die Arbeitsgruppen mit der eigentlichen Ausarbeitung der Inhalte. Beide Gruppenarten setzen sich aus Mitgliedern des W3C zusammen. In den Arbeitsgruppen sitzen also auch viele Vertreter von Softwareherstellern. Das unabhängige Kern-Team des W3-Konsortiums überwacht die Aktivitäten. Beim Entwickeln von Standards besteht die Ausarbeitung im Erstellen und Publizieren von technischer Referenz-Dokumentation zu dem betreffenden Standard.

Über die Einstiegsseite des W3-Konsortiums ([www.w3c.org/](http://www.w3c.org/)) können Sie alle Aktivitäten der Organisation mitverfolgen. Mittlerweile gibt es auch ein Deutsches Büro des W3-Konsortiums, das in deutscher Sprache über die Arbeit des W3-Konsortiums berichtet ([www.w3.org/Consortium/Offices/Germany/](http://www.w3.org/Consortium/Offices/Germany/)). Für alle, die sich professionell mit Web-Design beschäftigen, sind regelmäßige Besuche auf den Web-Seiten des W3-Konsortiums mittlerweile Pflicht.

## 1.5 Hypertext

### 1.5.1 Definitionen zu »Hypertext«

#### ***Hypertext und Text***

Text wird »der Reihe nach« geschrieben und gelesen. Diese Linearität liegt jedoch nicht an der Natur des Mediums »Text«, sondern an den bisher üblichen Text-Präsentationsmedien. Angefangen von der Papyrus-Rolle bis hin zum modernen Buch: Text wird wie selbstverständlich als Buchstabenwurm an einem langen Spieß aufgereiht. Frederic Vester beschreibt dies am Ende seines Buchs *Neuland des Denkens* folgendermaßen: »Das Medium »Buch« mit der linearen Anordnung eines in diesem Falle ca. 1,9 Kilometer langen Buchstabenwurms ist, wie letztlich die Sprache überhaupt, nur bedingt geeignet für das Thema, das ich mir hier vorgenommen habe. Der Versuch, ein Gesamtbild der vernetzten Wirklichkeit zu geben, die eigentlich nur simultan erfasst werden kann, muss daher zwangsläufig unvollkommen bleiben.«

Ohne auf die metaphysische Verzweiflung, die in diesem Zitat zum Ausdruck kommt, näher einzugehen, lässt sich feststellen: Es gibt eine moderne Kritik an der Linearität. Nicht unbedingt an der Linearität auf der Ebene einzelner Sätze oder Passagen, sondern auf der Ebene eines größeren Textes, der aus vielen, in sich erkennbaren Teilen besteht und aufgrund seines Präsentationsmediums dazu verdammt ist, von vorne bis hinten wahrgenommen zu werden.

Hypertext ist ein nichtlineares Medium zur Präsentation von solchen Texten. An die Stelle des Buchstabenwurms treten vernetzte Einheiten von Text. Durch die Unterstützung der Computer schwindet die Abhängigkeit von der Rolle, vom rein linearen Medium. Bei Weiterdenken hat das die Hoffnung ausgelöst, man könne das Verfassen und Lesen von Texten dem menschlichen Denken und Verstehen besser anpassen als bisher.

### ***Hypertext, Hypermedia und Multimedia***

Hypertext ist nicht etwas anderes als Text. Hypertext ist nach wie vor Text, nur kein Text mehr, der dem Leser durch sein reines Erscheinungsbild suggeriert, man müsse ihn von vorne bis hinten lesen. Vermittelt wird auch bei Hypertext Information, Unterhaltung usw. Unter den Medien geht der Trend allerdings hin zu visuell auffälligeren Medien, also zu Bildern, Videos, und simulierten, bewegten Welten. Die konsequente Weiterentwicklung von Hypertext ist deshalb »Hypermedia«. Hypermedia ist jedoch etwas anderes als »Multimedia«, das in aller Munde ist. »Multimedia« heißt nichts anderes als: »viele Medien zugleich«. »Hypermedia« bedeutet dagegen: »viele Medien, jedoch in einer Weise organisiert, wie es von Hypertext her bekannt ist«. Das Präfix »Hyper-« steht also für eine bestimmte Art von Organisation. Diese Art von Organisation zeichnet sich aus durch Fragmentierung einzelner Einheiten einerseits und intelligente Vernetzung dieser Einheiten andererseits.

Nicht der Unterschied zwischen »Hypertext« und »Hypermedia« ist entscheidend, sondern der Unterschied zwischen »Hyper-« und »Multi-«. Alles auf einmal serviert zu bekommen, führt lediglich zu Reizüberflutung. Entscheidend ist für den Rezipienten, alles so serviert zu bekommen, wie er es sich jeweils wünscht. Dieses Versprechen versuchen Hypertext und Hypermedia einzulösen.

### ***Hypertext, Kapitel, Verzeichnisse und Querverweise***

Stark strukturierte Bücher enthalten im Prinzip alles, was Hypertexte auch enthalten: hierarchische Strukturen (Kapitel und Unterkapitel), verbindende Elemente (Querverweise) sowie unterschiedliche Informationszugänge (Inhaltsverzeichnisse, Stichwortverzeichnisse). Der Unterschied liegt allein in der Präsentation: das Buch »empfiehlt« sich rein äußerlich trotz der nicht-linearen Angebote letztlich, linear bewältigt zu werden – »Seite für Seite«. Bei Hypertexten, die am Bildschirm präsentiert werden, wird diese stillschweigende »Recommendation« von Büchern aufgebrochen.

Überspitzt kann man es so ausdrücken: Text in Büchern, auch in stark strukturierten Büchern, liest man; Hypertext dagegen liest man nicht, sondern man »navigiert« darin. Ob das Lesen dabei tatsächlich zu kurz kommt, ist eine durchaus ernst zu nehmende Frage, der sich das Medium Hypertext stellen muss.

Es gibt durchaus auch Bücher, die nicht von vorne nach hinten gelesen werden: Lexika beispielsweise, oder auch Fachbücher mit Nachschlage-Charakter. Und genau bei dieser Sorte Literatur hat sich Hypertext bislang auch am erfolgreichsten bewährt. Kaum vorstellbar ist Hypertext dagegen bei literarischen Gattungen wie klassischen, aus einem klaren Handlungsstrang bestehenden Romanen. Andererseits besitzen die meisten modernen Romane eben keinen solchen klaren Handlungsstrang mehr. Seit mindestens hundert Jahren schon bewegt sich auch die schöngestige Literatur weg von der Linearität. Damit wird sie ebenfalls relevant für Hypertext, auch wenn sich auf diesem Gebiet alles eher noch in der Experimentierphase befindet.

### ***Hypertext, Computer und Computernetze***

Hypertext wird heute ausschließlich auf Computern realisiert, weil Computer die spezifischen Anforderungen von Hypertext hervorragend unterstützen. In der Vergangenheit gab es jedoch auch Versuche, so etwas wie Hypertext mit herkömmlichen Mitteln zu realisieren. So kam in den

frühen siebziger Jahren ein didaktisches Konzept in Mode, das sich »programmierte Unterweisung« nannte. Es handelte sich um Bücher, die man nicht sequenziell lesen konnte, sondern die so aufgebaut waren, dass der Schüler kleine Informationshäppchen oder Fragen angeboten bekam und dazu eine Auswahl an Querverweisen zu anderen Stellen im Buch. Auch die Verweisziele waren Informationshäppchen oder Kommentare zu der ausgewählten Antwort mit neuen Querverweisangeboten zu anderen Stellen.

Es zeigte sich jedoch, dass das viele Blättern in solchen Büchern nicht gerade die Konzentration auf den Lehrstoff förderte. Hier sind Computer weit überlegen. Ein Klick mit der Maus auf einen Verweis, und das gewünschte Verweisziel wird angezeigt. Diese Tatsache kann gar nicht unterschätzt werden. Unterbrechungszeiten von mehr als drei Sekunden zerstören nachweislich eine anhaltende Konzentration. Blättern dauert meist länger und erfordert außerdem Gehirn-Reserven. Klicken und warten, bis der neue Inhalt da ist, geht in der Regel schneller und verbraucht keine oder viel weniger Reserven.

Elektronische Präsentation von Text hat allerdings auch Nachteile (es geht hier nicht um die Zwangsvorstellungen von Leuten, die meinen, nicht ohne Papier leben zu können): einem elektronischen Hypertext kann man nicht ansehen, wie groß er ist, d.h. man kann sich keine rechte Vorstellung davon machen. Das kann beängstigend oder beruhigend wirken, je nachdem, wie der Anwender gegenüber dem Medium eingestellt ist. Es kann angenehm sein, nicht zu wissen, wie groß »das Ganze« ist, weil man ja zunächst sowie nicht alles zu überblicken imstande ist. Es kann aber auch unangenehm sein, so gar keine Vorstellung über das Ganze zu haben. Hypertext-interne Techniken wie Web-Views versuchen entsprechenden Ängsten beim Anwender entgegenzutreten.

Aber es bleibt nicht bei dem Problem, die Größe eines einzelnen Hypertextes zu erfassen. Hypertexte neigen nämlich ferner dazu, sich untereinander zu vernetzen und dadurch immer größere »Ganzheiten« hervorzubringen. Deshalb sind lokal auf einem Computer oder Speichermedium verfügbare Hypertexte auf die Dauer unbefriedigend. Schon früh (bereits 1965 bei Ted Nelson) kam deshalb der Wunsch auf, Hypertext-Einzelprojekte weltweit zu vernetzen. Doch erst Ende der achtziger Jahre, als das Thema Hypertext in weiteren Kreisen diskutiert wurde, wurde die Idee geboren, das weltweite Internet für einen Hypertext-orientierten Dienst zu nutzen. Aus diesen Überlegungen entstand Anfang der neunziger Jahre schließlich das World Wide Web.

### **Grundsätzliche Positionen zu Hypertext**

Folgende grundsätzliche Positionen zu »Hypertext« lassen sich unterscheiden:

- Jene Position, die in Hypertext die konsequente Weiterentwicklung von stark strukturierten Texten sieht. Zitat Kühlen: *»Wir kommen zu dem Ergebnis, dass Hypertext die auch schon in Texten verwendeten nicht-linearen Gestaltungsmittel zum Prinzip erhebt...«*. (Rainer Kühlen: Hypertext – Ein nicht-lineares Medium zwischen Buch und Wissensbank, Berlin, Heidelberg, New York, 1991)
- Jene Position, die in Hypertext eine neue, besonders benutzerfreundliche Recherche in großen Informations- und Datenbeständen sieht. Zitat Schnupp: *»Hypertext spannt über Textdokumente eine Hyperstruktur zur Unterstützung von Recherchen auf«*. (Peter Schnupp: Hypertext; In der Reihe: Handbuch der Informatik, Band 10.1, München, 1992)



- Jene Position, die in Hypertext die Schaffung neuer kreativer Freiräume und Dimensionen für Autoren und Leser sowie das Verschwimmen der Grenzen zwischen Autor und Leser sieht. Zitat Joyce: *»Hypertext narratives become virtual storytellers and narrative is no longer disseminated irreversibly from singer to listener or writer to reader.«* (Hypertext-Erzählungen werden virtuelle Geschichtenerzähler, und Erzähltes wird nicht mehr unumkehrbar nur von Sänger zu Zuhörer oder von Autor zu Leser verbreitet). (Michael Joyce: Interview, Discover, The World of science, November 1989)

Rainer Kuhlen vertritt die erste Position am deutlichsten. Über die Linearität von Text schreibt er: *»Linearität beruht aus texttheoretischer Sicht auch auf der richtigen Verwendung mikrotextueller, Kohäsion erzeugender Mittel«*. Dazu gehören z.B. ausdrückliche Leseanweisungen wie »dies ist so zu verstehen, dass...«, oder hinweisende Ausdrücke wie »dies wurde ja schon im ersten Kapitel problematisiert«. Die Zusammenhänglichkeit linearer Texte benutzt also immer auch Mittel, welche die Linearität sprengen. Diese Linearität sprengenden Mittel werden in Hypertexten einfach zum Prinzip erhoben. Aus Texten mit textimmanenten, Kohäsion erzeugenden Mitteln werden »informationelle Einheiten« (Kuhlen), die durch ausführbare Verweise untereinander vernetzt werden. Die Verweise leisten dabei nichts grundsätzlich Neues. Es sind keine willkürlichen, sondern semantisch begründbare, argumentative Verknüpfungen, die zur Navigation oder zum Herstellen von Querbezügen dienen. Neu ist nur deren computerunterstützte Ausführbarkeit.

Im Web gibt es beispielsweise unzählige kleine, in sich abgeschlossene Projekte, die bewusst die Mittel von HTML nutzen, um eine Hypertext-Struktur zu realisieren (die Homepage, bestehend aus ein paar Web-Seiten). Solche Projekte sind jedoch zu klein, um darin umfangreiche Daten-Recherchen durchzuführen. Sie sind am ehesten durch die Position zu beschreiben, die Rainer Kuhlen einnimmt.

P. Schnupp betont dagegen die Recherche-Position am deutlichsten – wohl auch deshalb, weil er kleinen Projekten gar keinen Hypertext-Charakter zubilligt. Seine Einführung zum Thema beginnt mit den Worten: *»Das Bedürfnis, aus dem Hypertext entstand, ist das gleiche, welches von der Datei zur Datenbank führte«*. So wie die Datenbank bei strukturierten Daten gegenüber der Einzeldatei neue Möglichkeiten der Datenabfrage erschloss, will Hypertext *»die interaktive Recherche in umfangreichen Textbeständen erleichtern und beschleunigen«*.

Neu sind für Schnupp bei Hypertext vor allem die intuitiven, kein tieferes EDV-Wissen voraussetzenden Recherche-Techniken: *»Ähnlich wie die Spreadsheets dem individuellen Benutzer die Programmierung numerischer Anwendungsaufgaben zugänglich machten, bringt Hypertext ihm die selbständige Erstellung und Pflege umfangreicher Textarchive. Wie bei Daten wird auch bei Texten erst durch deren Verknüpfung ein großer Teil des in ihnen gespeicherten 'Wissens' zugänglich«*.

Bei Projekten mit großen Textbeständen stößt eine Realisierung mit lauter einzelnen, statisch abgespeicherten und aufrufbaren Texteinheiten in der Tat irgendwann an Grenzen. Bei solchen Datenbeständen erwartet der Anwender zu Recht eine Suchmöglichkeit, die den Datenbestand nach Ausdrücken oder Kriterien durchforstet und die Suchergebnisse »on demand« aufbereitet.

Die Auffassung vom kreativen Grundwesen von Hypertext hält sich ebenfalls noch immer. Gerade im literarischen Bereich wird – zumindest dort, wo die Autoren mit Computern umgehen können – mittlerweile viel mit Hypertext experimentiert. Dabei steht jedoch nicht unbedingt die Transparenz bei der Informationsbeschaffung im Vordergrund, sondern im Gegenteil sogar eher das »Locken des Lesers in lauter interessante Teilwelten«. Der Rezipient wird dabei bewusst

dem Stress ausgesetzt, an einer anderen Stelle, die er bereits verfolgt, aber per Verweis wieder verlassen hat, etwas zu verpassen. Ob dieses bewusste Unter-Stress-gesetzt-werden allerdings von Rezipienten als spannend oder eher als nervig empfunden wird, gehört zur künstlerischen Kritik. Hypertext ist jedenfalls in der Lage dazu, solche Dinge mit Lesern zu treiben, und vernetzte Texte, die für sich selbst in Anspruch nehmen, ein künstlerischer Ausdruckswille zu sein, lassen sich nicht mit informationswissenschaftlichen Methoden messen.

## 1.5.2 Begriffe rund um Hypertext

### *Nodes, Units, Items, Informationelle Einheiten*

Hypertexte sind so konzipiert, dass sie nicht zum sequenziellen Lesen gedacht sind. Was aber soll das sequenzielle Lesen ersetzen? Die Antwort: Texteinheiten, die in sich abgeschlossen sind und jeweils einen kleinen »Informationshappen« anbieten. Durch die intelligente Vernetzung solcher Texteinheiten entsteht dann jener Informationsraum, den man als Hypertext bezeichnet. Für die Texteinheiten hat man in der Vergangenheit viele Namen gefunden. Hier eine Auswahl:

- **node** (beim Datenmodell HAM)
- **unit** oder **item**
- **document** (beim Softwareprodukt Intermedia)
- **card** (bei den Softwareprodukten NoteCards und HyperCard)
- **information blocks**
- **frame** (beim Softwareprodukt KMS)
- **statement** (beim Softwareprodukt Augment)
- **message** (beim Softwareprodukt KnowledgePRO)
- **article** (beim Softwareprodukt HyperTies)
- **hyper-molecule** (beim Softwareprodukt HyperCOSTOC)
- **guideline** (bei den Softwareprodukten Guide und NaviTextSAM)

Einige der genannten Softwareprodukte sind historisch und nicht mehr auf dem Markt. Die Liste lässt jedoch erkennen, dass all diese Produkte so etwas wie informationelle Einheiten kennen. Nur das World Wide Web lässt etwas Derartiges vermissen, und auch in HTML gibt es nichts, wodurch sich eine informationelle Einheit definieren lässt. Für diese Zurückhaltung gibt es allerdings gute Gründe. Das Definieren informationeller Einheiten kann nämlich leicht zu Einschränkungen in Bezug auf deren Inhalt führen. Eine »card« oder eine »message« ist im Grunde kein leerer Behälter, sondern ein Behälter, der schon in seinem Namen, aber auch meist durch seine technische Umsetzung gewisse Vorstellungen von dem ausdrückt, was er als Inhalt erwartet. Auf solche Vorgaben wollte man im Web verzichten. In der Regel ist eine HTML-Datei eine sinnvolle informationelle Einheit. Manchmal sind es aber auch Teile einer HTML-Datei, die per Anker anspringbar sind. Auch kann ein ganzes, überschaubares und in sich geschlossenes Projekt, das aus mehreren Dateien besteht, als informationelle Einheit betrachtet werden. Bei dynamisch erzeugten Inhalten sind die Grenzen einer Einheit wieder andere. Im Web ist auch Text möglich, der gar nicht wie »typischer Hypertext« wirkt. Auch die Dokumentation, die Sie hier lesen, hat zum Teil eher klassisch-sequenziellen Lese-Charakter als das, was man von typi-

schem Hypertext erwartet. Dennoch gibt es auch in dieser Dokumentation und im Web allgemein zweifellos einzelne, erkennbare Topics. Es gibt nur kein vorgegebenes Schema, in das sich solche Topics einpassen müssen.

### ***n:m-Relation, kohäsive Geschlossenheit***

Eine Hypertext-Einheit muss dem Anspruch genügen, von ganz verschiedenen Kontexten aus gesehen ein sinnvolles Verweisziel zu sein. Wenn der Anwender an irgendeiner anderen Stelle auf einen Verweis klickt, der zu dieser Texteinheit führt, muss die Texteinheit so gestaltet sein, dass sie in jedem Fall den Gedankensprung, den der Anwender mit dem Verweis »gewagt« hat, belohnt. Da der Kontext, also der Informationszusammenhang, von dem aus der Anwender zu dieser Texteinheit springt, ganz verschieden sein kann, muss diese Texteinheit so konzipiert sein, dass sie alle denkbaren Kontexte befriedigt. Das kann sie nur leisten, indem sie sich gewissermaßen »einigelt« und sich radikal auf ihre eigene Aussage konzentriert. Diese Einigelung bezeichnet man als »kohäsive Geschlossenheit«. Die Konstellation, dass viele verschiedene andere Texteinheiten zu ein und derselben Texteinheit verweisen können, gilt als n:m-Relation (n = viele, m = irgendein Ziel, aber ein bestimmtes Ziel).

Mit dem Begriff der »informationellen Einheit« versucht Kuhlen, einen informationswissenschaftlich diskutierten Begriff für die »Einheiten« eines Hypertextes zu etablieren. Dabei geht er auch auf die Problematik der kohäsiven Geschlossenheit solcher Hypertext-Einheiten ein, denn:

*»Wissen bzw. aktuelle Information ist immer in einen größeren semantischen und/oder argumentativen Kontext eingebunden« (Kuhlen, S.81). Es gibt kein »autonomes Stück Information«. Wie kann dann in Hypertext-Einheiten, die dem Anspruch genügen müssen, kohäsiv geschlossen zu sein, Information angeboten werden? »Obgleich die Kunst zweifellos darin besteht, informationelle Einheiten so festzulegen, dass sie aus sich heraus verstanden werden können, müssen sie doch gleichzeitig auf ihren informationellen Kontext verweisen« (Kuhlen, S.81).*

Eine Hypertext-Einheit muss demzufolge inhaltlich in sich geschlossen sein und gleichzeitig ausdrücklich auf übergeordnete oder verwandte Information verweisen, um dem Leser/Anwender so ihren Informationskontext zu zeigen. Dies kann durch ausführbare Querverweise, durch Verweise auf übergeordnete Ebenen oder durch Verweise auf besondere Navigationshilfen geschehen.

### ***Web-Views***

Einem elektronisch gespeicherten Hypertext kann man – im Gegensatz zu einem Buch etwa – nicht unmittelbar ansehen, wie umfangreich er ist. Der Leser/Anwender tut sich mit der Navigation im Informationsangebot jedoch leichter, wenn er sich eine plastische Vorstellung des Informationsangebots machen, sich also einen »Informationsraum« mit dieser oder jener Größe und Komplexität vorstellen kann. Das verringert die Gefahr des »Lost in Hyperspace«-Gefühls (siehe den gleichnamigen Abschnitt weiter hinten).

Web-Views sind Gesamtsichten auf einen Hypertext mit der Absicht, den dargebotenen Informationsraum zu visualisieren. Ein Web-View soll dem navigierenden Anwender den gesamten Hypertext als eine Art Stadtplan zeigen und möglichst in jeder beliebigen Situation ein »Sie befinden sich hier!« ausgeben können. Dazu bedarf es allerdings ausgereifter grafischer Darstellungsmöglichkeiten. Man stelle sich einen Hypertext mit zigtausend Texteinheiten und noch

mehr Verknüpfungen vor. Die Visualisierung eines solchen Netzes wirft Probleme auf, und es ist fraglich, wie groß der Orientierungsgewinn für den Navigierenden dabei ist. Sinnvoller sind in der Praxis lokale Web-Views, die den aktuellen Dokumentteil im Zusammenhang darstellen, oder ferne Web-Views, die vor dem Ausführen eines Verweises zu einem anderen Dokumentteil diesen Dokumentteil auf Wunsch in Form einer Übersicht präsentieren.

### ***Fish-Eye-Views***

Ein Fish-Eye-View ist das Angebot, per Verweis zu näherer Information über ein angeschnittenes Thema zu gelangen. Die Möglichkeit, sich in ein beliebiges Thema unverzüglich und beliebig weit vertiefen zu können, gehört zu den Hypertext-Visionen. Kein langes Blättern, kein Griff in ein anderes Regal, kein Gehen in eine andere Bibliothek mehr, sondern nur ein Mausklick oder ein Tastendruck. Verfügbarkeit der Information im Moment der Konzentration und des spontanen Interesses. Das soll den »informationellen Mehrwert« (Kuhlen) von Hypertext gegenüber herkömmlicher Informationsbeschaffung begründen.

Voraussetzung sind Hypertext-Netze von gigantische Ausmaß. Als der große Vordenker solcher Netze gilt Ted Nelson. Mit seinem Projekt **Xanadu** hat er solch ein Netz schaffen wollen. Der Durchbruch wurde dem Projekt jedoch verwehrt. Stattdessen haben wir heute mit dem World Wide Web ein Netz, dessen Wachstum selbst die kühnsten Träume der Hypertext-Visionäre in den Schatten stellt. Doch gerade das unkontrollierte Wuchern dieses Netzes macht es schwer, Fish-Eye-Views zu realisieren. Da hier auch die Qualität der Information so unterschiedlich ist, ist ein organisiertes System von Fish-Eye-Views allenfalls in Teilen des WWW realisierbar, etwa zwischen bestimmten wissenschaftlich orientierten Informationsbereitstellern.

### ***Information Retrieval***

Information Retrieval ist die Art und Weise, wie ein Anwender in einem großen Datenbestand letztendlich an konkrete Informationen herankommt. Ein ausgereiftes Hypertext-Projekt erreicht vom Umfang her leicht vielbändige Werke. Ein solcher Hypertext wird wohl eher zur gezielten Informationssuche genutzt, nicht zum gemütlichen Schmökern. In klassischen Datenbanken gelangt der Informationssuchende zu seiner gewünschten Information über entsprechende Suchabfragen. Diese Suchabfragen muss er formulieren, z.B. »zeige mir alle Kunden, die in Monaco wohnen, ledig sind und mehr als 10000 Mark im Monat verdienen«. Dabei muss man jedoch vorher wissen, wonach man sucht. Bei textorientierten Informationsbeständen ist die Formulierung einer Suchabfrage schwieriger als bei strukturierten Daten. Stichwortsuchen in einer Volltextdatenbank können – zumindest ohne Einsatz von künstlicher Intelligenz – niemals so aufschlussreiche und exakte Suchergebnisse liefern wie Suchabfragen in einer feldorientierten Datenbank.

In textorientierten Informationsbeständen muss der Informationszugang deshalb idealerweise mehrgleisig organisiert werden. Die programmunterstützte Stichwortsuche sollte ebenso möglich sein wie der Zugang über ein fein gegliedertes System von Verweisen.

### ***Orientierung, Navigation, »Lost in Hyperspace«***

Wenn man einen Hypertext »aus der Vogelperspektive« betrachtet, gibt es darin n:m-Relationen. Aus Sicht des Lesers eines Hypertextes gibt es jedoch keine n:m-Relationen. In einer gegebenen

Situation gibt es höchstens eine 1:n-Relation, also mehrere Verweise, die von der aktuellen Texteinheit wegführen. Diese Verweise stellen – gemeinsam mit immer verfügbaren Hypertext-Funktionen wie Backtracking und Lesezeichen – das Arsenal an Orientierungsmöglichkeiten dar, die der Anwender nutzen kann, um sich im Informationsangebot sinnvoll zu bewegen.

Was sich aus der Vogelperspektive als freies Bewegen in einem faszinierenden Netz darstellen mag, ist aus der Sicht eines konkreten Benutzers dieses Netzes die Aufgabe, sich in dem Netz, das er nicht unmittelbar überschaut, zurechtzufinden, zu orientieren. Sich in einem nicht komplett übersehbaren Hypertext-Netz bewegen heißt, darin zu **navigieren**. Die Anleihe aus der Welt der Hochseeschifffahrt ist bewusst gewählt und hat den Browser-Anbieter Netscape dazu bewegt, sein berühmtes Programm zunächst *Netscape Navigator* zu nennen.

Der Navigierende orientiert sich an den angebotenen Verweisen. Wenn die Verweise nicht intelligent sind, keinen kontextuellen Zusammenhang herstellen, droht der Navigierende die Orientierung zu verlieren und in jenen Zustand zu geraten, den man in der Hypertextliteratur als **Lost in Hyperspace** – verloren im Hyperraum – bezeichnet.

### ***Pfade, Trails, Guided Tours***

Die meisten Hypertext-Anwender sind noch keine erfahrenden »Navigatoren«. Sie sind die Informationsaufnahme von Büchern gewöhnt und wünschen sich neben den waschechten Hypertext-Sprüngen auch die Möglichkeit, das Informationsangebot nach einer vorgegebenen Schrittfolge durchzugehen. Solche Ketten von aneinandergereihten Einheiten eines Hypertextes bezeichnet man als **Pfad** – engl. *trail* – oder auch als **geführte Touren** – engl. *guided tours*.

Die HTML-Version dieses Buches bietet Ihnen auf Seiten wie dieser hier beispielsweise am Seitenende Sprünge zur nächsten und vorhergehenden Seite an. Dabei ist es gewiss nicht immer so, dass das nächstliegende Thema dem aktuellen Thema inhaltlich besonders nah steht. Es handelt sich vielmehr um ein Angebot, das auch in kritischen Situationen zum Weiter Navigieren ermutigen soll – also in Situationen von der Art: »Ich weiß ja gar nicht, was mich als Nächstes interessiert« (das typische Gefühl des Lost in Hyperspace).

### ***Backtracking, Historie, Lesezeichen***

**Backtracking** meint das Rückwärtsverfolgen eines eingeschlagenen Lesepfads. **Historie** ist das, was dazu nötig ist: ein Protokoll der bis zu einem Zeitpunkt besuchten Hypertext-Einheiten. **Lesezeichen** – engl. *bookmarks* – sind Marken, die der Leser an für ihn besonders interessanten oder zentralen Texteinheiten setzt und zu denen er jederzeit springen kann.

Im Hypertext selbst sind solche Funktionen nicht realisierbar, höchstens ansprechbar (so kann der Autor einer Web-Seite mit JavaScript beispielsweise die besuchten Web-Seiten des Anwenders abfragen). Zunächst einmal muss jedoch die verwendete Navigations-Software diese Funktionen dem Anwender zur Verfügung stellen, da es sich in beiden Fällen um sessionspezifische Navigationsdaten bzw. anwenderspezifische Navigationsschwerpunkte handelt.

Bei einigen Softwareprodukten ist die Lebensdauer von Lesezeichen und Historie auf eine Sitzung (Session) begrenzt, d.h., die Daten werden nur im Arbeitsspeicher gehalten und gehen nach Beenden des Navigationsprogramms verloren, oder sie sind dauerhaft in Dateiform speicherbar. Die meisten modernen Browser erlauben dem Anwender, selber einzustellen, wie weit die Historie zurückverfolgbar sein soll.

### **Browsing, Authoring**

**Browsing** bedeutet Herumstöbern, und zwar mit kreativem oder informationellem Mehrwert. Das ist es, was sich begeisterte Hypertext-Anhänger für den Leser eines Hypertextes erhoffen. Doch solange es noch keine Hyper-Computer gibt, werden die Verweise, die der Leser entdecken soll, von Autoren eingerichtet.

**Authoring** ist der eigentlich unmögliche Versuch, einen Hypertext mit all der Intelligenz auszustatten, die dem Leser den Mehrwert gegenüber sequenziellem Text erst vermitteln soll. Eigentlich müssten also hyper-intelligente Autoren für normal-intelligente Leser schreiben. Doch nicht nur, dass normal-intelligente Autoren an Hypertexten schreiben – nein, auch Leser selbst schreiben Hypertexte fort. Denn auch das Fortschreiben durch Anwender gehört zum Authoring in manchen Hypertext-Systemen. Die Trennung zwischen Autor und Rezipient ist im Hypertext-Bereich zumindest theoretisch fließend und zum Teil aufgehoben, weil printmedienbedingte Zwänge wie Auflagenhöhen, Neuauflagenkalkulationen usw. weitgehend entfallen.

Bei Hypertext-Systemen unterscheidet man generell zwischen reinen Browsern (Nur-Lese-Zugriff) und Systemen, die das Editieren bzw. Fortschreiben erlauben. Ein Fortschreiben von allem seitens aller ist jedoch in der Praxis nur in geschlossenen Benutzerkreisen möglich, um rechtlichen Missbrauch von geistigem Eigentum zu vermeiden und Qualität zu wahren. Im Web herrscht deshalb – obwohl Gründervater Tim Berners-Lee sich dagegen lange wehrte – eine klare Trennung zwischen Anbietern und Lesern von Web-Seiten. Über technische Umwege wie die CGI-Schnittstelle ist es allerdings möglich, interaktiv wachsende Web-Projekte, an denen mehrere Personen online arbeiten können, zu realisieren. Ein Diskussionsforum ist beispielsweise eine derartige Anwendung, und Diskussionsforen mit vielen, hochwertigen Beiträgen gehören deshalb wohl auch zu den spannendsten, modernen Formen von Hypertext.

### **Sharks, Holy Scrollers, Karteisysteme, Buchmetapher**

In der Hypertext-Fachwelt geht ein Streit zwischen jenen, die das Scrollen (Hoch- oder Runterfahren im Text) verboten haben wollen, und solchen, die es erlauben möchten. **Sharks** (*Haie*) nennt man jene, die dafür plädieren, dass eine Texteinheit immer auf einen Blick erfassbar sein sollte; **Holy Scrollers** sind jene, die dem Umfang einer Texteinheit keine bildschirmbedingten Grenzen aufzwingen wollen. Sharks tendieren dazu, Hypertexte als Karteisysteme zu kreieren. Grundelement ist dabei die Karteikarte mit ihrer unveränderlichen Größe. Holy Scrollers bevorzugen dagegen die Buchmetapher beim Design von Hypertexten. Grundelement sind dabei Textabschnitte von variabler, den Gegebenheiten angepasster Länge.

Heute haben sich weitgehend Systeme durchgesetzt, die einer Texteinheit keine Längenbeschränkungen auferlegen. Auch Web-Seiten können beliebig lang sein. Dennoch wird gerade im WWW mit Hilfe von Techniken wie Frames häufig auch eine scroll-freie Informationsdarbietung angestrebt, denn das Erfassen eines gesamten Informationszusammenhangs auf einen Blick hat unbestreitbare Vorteile.

### **Hypertext Abstract Machines**

»In vielen Bereichen der Programmierung hat es sich oft als die beste Implementierungsmethode erwiesen, mit softwaretechnischen Mitteln eine abstrakte oder virtuelle Maschine...für eine Klasse von

*Anwendungsprogrammen zu realisieren, auf der dann die eigentliche Software aufbaut.» (Schnupp, S.85).*

Ein Beispiel dafür sind Datenbanken wie *dBase*, *FoxPro* u.v.a., die an die Stelle spezieller Einzelsoftware treten, etwa für Kundenverwaltung, Bibliotheksbestand-Verwaltung usw. Für Hypertext gilt das Gleiche: Man kann einen Hypertext fest programmieren, oder man programmiert eine unabhängige, abstrakte Software, die das Implementieren von Hypertexten unterstützt, indem sie geeignete Mittel und Funktionen zur Verfügung stellt. Als professionelle Systeme dieser Art nennt Schnupp z.B. das Produkt *HAM* (Hypertext Abstract Machine) oder *HyperBase*.

Noch einen Schritt weiter geht jedoch das Konzept, lediglich ein Hypertext-Datenformat zu standardisieren, ohne es an eine bestimmte Software zu binden. Das bisher anspruchvollste Format dieser Art ist SGML (Structured Generalized Markup Language), heute eher in seiner moderateren Ableitung XML verbreitet. Eine Unterklasse von SGML und neuerdings auch von XML ist HTML, das Datenformat für Hypertext im World Wide Web. Hierbei handelt es sich genau genommen nicht mehr um eine Hypertext Abstract Machine, sondern um eine Hypertext Abstract Form – es ist nicht nur anwendungsungebunden, sondern auch maschinenungebunden.

### **World Wide Web**

Hypertext im Web geht in zwei Punkten über die an bestimmte Softwareprodukte gebundenen Hypertext-Realisierungen in der Geschichte von Hypertext (Kapitel 1.5.3) hinaus:

- Hypertext im Web setzt auf einer softwareunabhängigen Hypertext-Beschreibungssprache auf, nämlich auf HTML (Hypertext Markup Language).
- Hypertext im Web setzt auf einem weltweiten, bereits vorhandenen und etablierten Computernetzverbund auf, dem Internet.

Lediglich Ted Nelsons Xanadu-Projekt verfolgt einen ähnlich weitreichenden und globalen Ansatz, wurde jedoch von der Entwicklung des Web überholt. Das Web ist eine möglichst allgemein gehaltene Plattform für weltweiten Hypertext, indem es nur Formalien wie Datenformate, Adressierung und Software-Kommunikationsprotokolle festlegt.

Die Realität im heutigen Web sieht allerdings etwas anders aus als in den Träumen der weltweiten Informationsvernetzer. Es dominieren Kommerzialisierung, egozentrische Angebote und Null-Informationen. »Echter« Hypertext führt im Web eher ein Nischendasein. Doch solange die »echten« Hyper-Links ihren Weg durchs Netz finden, ist eine Mehrfachnutzung des Web nicht unbedingt schädlich. Die Kommerzialisierung inklusive Konkurrenzkampf kann auch dazu beitragen, dass finanziell problematische Aspekte wie höhere Bandbreiten bei der Datenübertragung in Zukunft leichter lösbar sind.

## **1.5.3 Geschichte von Hypertext**

### **Vannevar Bush und sein »Memex«**

*»Bushs 'Memex' ist ein (auf dem Papier entworfenes, nicht experimentell realisiertes) Informationssystem, das auf dem methodischen und technischen Stand dokumentarischen Wissens (Indexieren, kontrollierte Vokabularen, Relationierung und Mikroverfilmung) von 1945 aufsetzte, dieses*

*allerdings mit der Radikalisierung der Idee der Verknüpfung ('association') kritisierte und erweiterte und damit einen richtungsweisenden Entwurf vorlegte, der bis heute die Ideen von Systemgestaltern befruchtet. Memex war als eine Maschine konzipiert, welche das menschliche Gedächtnis und sein Assoziationsvermögen erweitern und unterstützen sollte...Bush stellte sich eine Kamera vor, die an der Stirn befestigt sein sollte und durch die alles, was im Verlaufe eines Tages das Interesse des Besitzers weckte, sofort aufgenommen und im Memex verfügbar gemacht werden sollte» (Kuhlen, S.67).*

Vannevar Bushs **Memex** (1945) gehört in den Bereich des technisch-wissenschaftlichen Utopismus. Doch die Memex-Utopie gehört zu jenen Utopien, deren Grundgedanke zu überzeugend ist, um sie einfach ins Reich der Fabeln zu verbannen:

*»The human mind...operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain« (Bush) – »der menschliche Geist ... arbeitet mit Hilfe von Verknüpfung. Gerade mal einen Inhalt in seiner Reichweite, schnappt er sich sogleich den nächsten, der von der Assoziation der Gedanken vorgeschlagen wird, entsprechend einem undurchsichtigen Netz aus Pfaden, die von den Gehirnzellen getragen werden.«.*

Für diesen Grundzug menschlichen Denkens und Verstehens stellte sich Bush eine adäquate maschinelle Unterstützung vor. Diese Vorstellung befruchtet bis heute die Begeisterung für Hypertext.

### **Douglas C. Engelbart und sein »Augment«**

**Augment** ist das Hypertext-System mit der längsten Vergangenheit. »Augment« wurde schon ab 1963 am SRI International in Stanford entwickelt, und ab 1978 wurden die kommerziellen Rechte an Tymshare übertragen.

*»AUGMENT wurde ursprünglich auf Time-sharing-Großrechnern in einer Netzumgebung entwickelt. Richtungsweisend wurden die eingesetzten Werkzeuge, die den heutigen Standards von Personalcomputern und individuellen Arbeitsplatzrechnern bestimmen: Maus als Eingabemedium, Mehrfenstertechnik mit 'paralleler' Verarbeitung, ... Verknüpfung heterogener Materialien über Zeiger, intensiver, integrierter Einsatz von Graphik...Engelbart sah die durch AUGMENT zu erzielenden Mehrwerte effekte gegenüber linearen Formen wie Bücher in aller Deutlichkeit... AUGMENT war von Anfang an in eine elektronische Kommunikationsumgebung eingebettet, die z.B. verteiltes Publizieren und Editieren, elektronische Dokumentauslieferung, elektronische Post, Telekonferenzen erlauben sollte.« (Kuhlen, S.69).*

Die meisten Hypertext-Designer der Vergangenheit unterschätzten im Gegensatz zu Engelbart die Bedeutung des »Look and Feel« für den Endanwender. Auch die Entstehung des World Wide Web hat gezeigt, dass der Durchbruch erst mit den modernen, für grafische Benutzeroberflächen verfügbaren Browsern gelang, nicht mit den Ideen und Konzepten allein, so überzeugend diese auch sein mochten. Engelbart dachte schon an grafische Oberflächen und Mäuse, als Steve Jobs und andere, die gemeinhin als die Begründer der grafischen Benutzeroberflächen gelten, noch Kinder waren. Doch die Zeit war einfach noch nicht reif für die grafischen Vorstellungen von Engelbart. Science-Fiction-Filme aus den 60er Jahren belegen nur allzu deutlich, wie sich die Menschen die Computer der Zukunft vorstellten: als schrankartige Gebilde, die zwar mühelos mit Menschen reden können, aber ansonsten nur aus Kontrolllämpchen, Knöpfchen und maximal einer Primitivkonsole mit einzeiligem Text-Display bestehen. Engelbarts Bedeutung für



die Geschichte von Hypertext besteht also vor allem in seiner Erkenntnis, dass Hypertext an visuell attraktive Darstellungstechniken und ergonomische Interaktionsformen gebunden ist.

### **Ted Nelson und sein »Xanadu«**

Die Arbeiten von Ted Nelson – besser, seine Visionen – sind das Fundament heutiger Hypertextforschung und -begeisterung. *»Nicht nur, dass die beiden Bezeichnungen 'Hypertext' und 'Hypermedia' auf ihn zurückgingen, sondern vor allem seine unkonventionellen und über Jahrzehnte verfolgten Ideen eines universalen Wissensverwaltungs- und Informationsbereitstellungssystems, XANADU, machten ihn zu einem der kreativsten Hypertextforscher. Bemerkenswert an Nelson ist weiterhin, dass er sich nicht an etablierte Institutionen und kommerzielle Firmen hat binden lassen, sondern eher immer alternative Wege gegangen ist, verbunden mit einem aufklärerischen Sendungsbewusstsein, das in den letzten Jahren allerdings durchaus die Chance für ökonomische Verwertung gesehen hat«* (Kuhlen, S.69). Nelsons Xanadu-Idee lässt sich bis ins Jahr 1965 zurückverfolgen. Seit Ende der 80er und dem Beginn der 90er Jahre wird »Xanadu« kommerziell vertrieben.

Nelson stellt sich unter Xanadu eine Informationsbank von unbegrenzter Größe vor. *»Das auch heute noch utopisch anmutende Endziel ist dabei die Verwaltung des gesamten Weltwissens über ein riesiges, computerunterstütztes Begriffsnetz, das den Zugriff auf die entsprechenden informationellen Einheiten gestattet. Durch die Möglichkeit der simultanen und kollektiven Bearbeitung eines Dokuments soll der tendenzielle Gegensatz zwischen Autor und Leser aufgehoben werden. Die Aufgabe solcher Systeme beschränkt sich nicht allein auf die Verwaltung der vielfältigen und komplexen Beziehungen zwischen einzelnen Hypertexteinheiten. Es sind darüber hinaus auch Mechanismen erforderlich, die bei extensivem Mehrbenutzerbetrieb die Integrität, Aktualität und auch das Rückverfolgen der Entstehungshistorie eines Dokuments gewährleisten sowie den Schutz von Urheber-, Nutzungs- und Vervielfältigungsrechten unterstützen* (Kuhlen, S.217). Nelsons Xanadu stellt für diese Aufgaben verschiedene Features zur Verfügung.

Über Workstations und leistungsfähige PCs ist der Online-Zugriff auf den Xanadu-Informationsbestand seit 1989/90 möglich. Das System erlaubt neben dem Informationsabruf auch das Einbinden und Vernetzen eigener Daten in den Informationsbestand. Dabei entscheidet der Autor, ob es sich um seine privaten Daten (Zugriff nur durch den Autor selbst möglich) oder um öffentliche Daten (Zugriff für alle möglich) handelt. Der Online-Zugriff auf den Informationsbestand ist kostenpflichtig. Autoren erhalten – computergesteuert – Tantiemen, wenn ihre Daten von anderen Benutzern über Verweise aufgerufen werden. Auf diese Weise kann jeder Benutzer zum Autor werden, und jeder Autor ist immer auch Benutzer. Xanadu hält ferner Werkzeuge zur Versionenkontrolle von Dokumenten bereit. Es bereitet die gespeicherten Daten für den Endbenutzer jedoch nicht selbst auf. Dazu ist eine »Front-End-Browsing-Software« nötig, die für die Bildschirmausgaben sorgt, die Verweise ausführt usw.

Damit hat Ted Nelson die Technologie des World Wide Web schon 25 Jahre vor dessen Entstehen vorweggenommen, und seine Konzepte sind zum Teil ausgereifter (z.B. das Tantiemensystem oder die Dokument-Versionenkontrolle). Verständlicherweise ist Ted Nelson erbost über den Siegeszug des Web, der einfach über ihn hinwegfegte.

### **Hypertext-Konferenzen**

Seit Ende der 80er Jahre ist »Hypertext« keine Sache einzelner, namhaft zu machender Pioniere mehr. Die Techniken von Hypertext wurden seit Mitte der 80er Jahre zunehmend in der Praxis

eingesetzt – vor allem in der Online-Dokumentation. Hypertext wurde zu einem anerkannten Gegenstand für Fachkonferenzen. Die ersten wichtigen Hypertextkonferenzen waren:

- 1. Hypertext-ACM-Workshop, 1987, University of North Carolina at Chapel Hill
- AI (Artificial Intelligence) and Hypertext, 1988
- Hypertext I, Aberdeen 1989
- Hypertext '89, Pittsburgh, PA
- Hypertext II, New York 1990
- ECHT (erste europäische Hypertext-Konferenz, Paris 1990)
- Hypertext/Hypermedia-Fachtagung, Basel 1990

### **Gegenwart und tiefere Gründe für Hypertext**

Die Ideen von Tim Berners-Lee, aus denen Anfang der 90er Jahre das World Wide Web entstand, sind also nicht im luftleeren Raum entstanden. Hypertext hat eine Tradition. Denn Hypertext ist eine Konsequenz aus der Vorstellungswelt, die sich im Laufe des 20. Jahrhunderts herausgebildet und alle Bereiche des modernen Alltags der Industriegesellschaften durchdrungen hat. Das moderne Leben darin ist gekennzeichnet durch Fragmentierung. Berufsleben, Familienleben, Vereinsleben erfordern verschiedene Rollen von ein und demselben Menschen und berühren sich nicht oder kaum. Der Broterwerb findet meist woanders statt als dort, wo die Früchte genossen werden. Auch für Erholung und Konsum sind Zonen entstanden, die eigens dafür aufgesucht werden. Der moderne Mensch hat gelernt, Dinge im Leben und im Kopf klar auseinanderzuhalten. Genau das aber ist das Prinzip der fragmentierten Einheiten, das auch Hypertext zugrunde liegt. Und Hypertext lässt sich als der Wunsch begreifen, diese fragmentierten Einheiten zumindest wieder sinnvoll zu verknüpfen, die Verbindungen zwischen ihnen sichtbar und ausführbar zu machen.

Man kann nur Brücken schlagen zwischen Ufern, die man auseinanderhält. Schon Friedrich Nietzsche sah menschliche Stärke darin, Gegensätze in sich auszuhalten. Der Mensch hat die Fragmentierung gewollt, um sich weiterzuentwickeln. Aber ebenso bleibt der Wunsch nach Einheit. Manche versuchen es mit Rückzug in einfache, traditionelle Lebensformen. Und andere eben damit, dem Gedanken der Vernetzung zu folgen. Sie spinnen das Netz («Weaving the Web«!).

## **1.5.4 Hypertext und menschliches Verstehen**

### **Vernetzte Texte für vernetztes Denken**

*»Hypertext scheint unter der Annahme kognitiv plausibel zu sein, dass Wissen im menschlichen Gehirn in vernetzt topologischen, nichtlinearen Strukturen organisiert sei. Unter dieser Annahme könnte die Wissensaufnahme über eine vergleichbare Organisationsform, wie sie durch Hypertext gegeben ist, effizienter sein als eine Aufnahme, die den 'Umweg' über lineare Präsentationsformen (Vorlesungen, Texte) nimmt« (Kuhlen, S.182).*

Wenn der Lernende, so die Argumentation, einen neuen Wissensgegenstand in einer vernetzten Umgebung kennen lernt, fällt ihm die Einordnung dieses Wissensgegenstandes in sein eigenes,

vernetztes Wissen leichter. Wenn z.B. von dem isolierten Wissensgegenstand »Quesenbandwurm« Verweise zu »Bandwürmer allgemein«, »Wirte und Zwischenwirte«, »Bandwürmer bei Menschen«, »Drehkrankheit« und »Krankheiten durch Bandwürmer allgemein« führen, helfen diese Verweise bei der Einordnung des »Quesenbandwurms« in das vorhandene Wissen.

### **Integrationskonflikt bei »Netz zu Netz«**

Gegen die Annahme, dass vernetzte Wissenspräsentation die Aufnahme von Wissen erleichtert, lassen sich zwei Argumente anführen:

- Das erste Gegenargument führt ins Feld, dass sich der Hypertext-Leser seine Leselinie erst selbst bahnen muss, was Zeit und Energie kostet. Wobei die Wissensaufnahme letztlich nicht anders funktioniert als bei »herkömmlichem« Text: sie *»findet..., auch durch die Navigation in Hypertext, in einer zeitlich sequenziellen Reihenfolge statt, sodass jeder faktische Weg letztlich doch wieder linear ist«* (Kuhlen, S.56) – siehe hierzu auch das Thema *n:m-Relationen und kohäsive Geschlossenheit* (Kapitel 1.5.2).
- Dem zweiten Gegenargument zufolge ist gerade die Integration von vernetztem Wissen in ein Wissensnetz problematischer als die Integration »einfachen« Wissens in das Wissensnetz. Dahinter steht die Annahme, *»dass zwei Netze, zumal wenn sie polyhierarchisch strukturiert sind, schwieriger zu integrieren sind als eine lineare Struktur in ein bestehendes Netz«* (Kuhlen, S.56).

### **Interaktion und Reaktionsspielraum**

Fein strukturierte Hypertexte bieten die Möglichkeit, Information nicht nur auf eine Art, sondern auf verschiedene Arten, über verschiedene Ebenen, in unterschiedlichen Zusammenhängen zu vermitteln. Der Hypertext-Leser, so die Argumentation, hat also einen Reaktionsspielraum bei der Informationsaufnahme. Durch Zoomen (Fish-Eye-Views), zwangloses Browsen (»Herumstöbern«), Web-Views usw. kann er in Interaktion mit dem Hypertext-System verschiedene Strategien ausprobieren, um bei Verstehensprozessen zu den gewünschten Aha-Effekten zu gelangen.

Denn jeder kennt den Effekt: Man versteht etwas nicht, obwohl man es ausführlich erklärt bekommt. Und plötzlich versteht man es – ohne viel Erklärung, einfach, weil man den gesuchten Bezugspunkt, das vermisste Stichwort, die individuell richtige Stelle zum Einordnen ins eigene Gedächtnis gefunden hat.

### **Unkontrollierte Informationsaufnahme**

Gegen Mehrschichtigkeit und Angebotsfülle von Hypertexten wird im Zusammenhang mit Wissenserwerb folgendes Gegenargument angebracht: Der Lernende befindet sich in einem *»anomalous state of knowledge«* (Brooks). In dieser unsicheren Situation ist das Angebot der freien Navigation gerade das Verkehrte. Gerade in diesem Zustand, in dem der Lernende den Stoff noch nicht überschaut, noch nicht viel versteht, noch nicht entscheiden kann, welche Information wichtig und welche unwichtig ist, muss das vermittelnde Medium (Lehrer, Professor, Buch) die Kontrolle übernehmen. Hypertext verlockt zu freiem Navigieren und damit zu Ablenkung, zu unkontrollierter Informationsaufnahme.

### »Vom Hundertsten ins Tausendste«: Vor- und Nachteile

»Vom Hundertsten ins Tausendste kommen« ist eine Redewendung mit negativer Bedeutung. Abstrahiert man diese jedoch, bedeutet die Redewendung einfach, dem freien Fluss der Gedanken zu folgen, ohne sich rationalen oder sonstigen Beschränkungen zu unterwerfen. Hypertext unterstützt diese Art zu denken. Ein Vorteil scheint diese Art zu denken in zwei Situationen zu sein:

- Zu Beginn der Beschäftigung mit einem neuen Gebiet. In diesem Stadium unterstützt die freie Navigation das Erwachen von Interesse und damit die geistige Bereitschaft, sich näher auf das Gebiet einzulassen.
- In einem fortgeschrittenen Stadium des Wissens. Wer sich auf einem Gebiet erst einmal auskennt, wird die Möglichkeiten von Hypertext gezielt und souverän nutzen können, da er weiß, wonach er sucht, wenn er ein solches System benutzt.

Ein Nachteil ist die von Hypertext favorisierte Art zu denken während der konzentrierten Wissensaufnahme, während des Lernstadiums, während jener Phase, in der sich der Aufnehmende mentale Modelle vom aufgenommenen Wissen macht. In dieser Phase sind Ablenkung und Angebotsfülle nur hinderlich. Um diese Phase der Informationsaufnahme elektronisch zu unterstützen, kann Hypertext allerdings die Technik der Guided Tours anbieten (Kapitel 1.5.2).

### Studien zu Hypertext

Die Schnotz-Studie (1987) untersucht die Wirkung von kontinuierlichen kontra diskontinuierlichen Texten auf den Lernerfolg. Das vermittelte Wissen war in beiden Fällen das gleiche. In den diskontinuierlichen Texten wurde der Textzusammenhang, d.h. alle argumentativen, verbindenden Strukturen entfernt. Die Studie, die an 34 Studenten durchgeführt wurde, ergab, dass die Gesamtinformation durch den kontinuierlichen Text tendenziell besser vermittelt wurde als durch die diskontinuierliche Version. Eine differenzierte Auswertung ergab: Personen mit höherem Vorwissen ziehen aus diskontinuierlichen Texten größeren Nutzen. Personen mit geringerem Vorwissen bevorzugen eindeutig kontinuierliche Texte. Die Schnotz-Studie war jedoch nicht als Vergleich von Hypertext und Text ausgelegt. Die Kohärenz und Kontinuität schaffende Leistung von Verweisen in Hypertexten wurde nicht berücksichtigt. Auf technische Dokumentation übertragen, könnte man sagen, dass die Studie eher den Unterschied zwischen Tutorial und Referenz untersucht hat. Die Intermedia-Studie (1987) bezieht sich auf zwei Kurse, einen über englische Literatur, und einen über Biomedizin. Zu beiden Kursen wurde das Lehrmaterial mit dem Softwareprodukt *Intermedia* als Hypertext organisiert. Die Ergebnisse der Studie waren keine Messungen des Lernerfolgs, sondern Auswertungen der Einschätzungen befragter Studenten. Die Hypertext-Unterstützung wurde insgesamt als wertvoller angesehen als die Unterstützung durch konventionelle Lehrmaterialien. Die Streuung zeigt allerdings, dass die Bewertungen von »much better« bis »much worse« reichen. Bei den Biomedizinern war die Resonanz positiver als bei den Literaturstudenten. Nach Aussagen der Dozenten wurden bei den Abschlussprüfungen deutlich höhere Leistungen erzielt. Weitere Studien zum Thema:

- Gordon u.a. präsentierten 1988 insgesamt 24 Studenten technische Texte am Bildschirm, und zwar jeweils in einer linearen und einer nicht-linearen Version. Tests zu den behaltenen Fakten ergaben eine Präferenz zugunsten der linearen Darbietung.
- McKnight/Dillon/Richardson präsentierten Versuchspersonen einen Text (»HYPTXTduction to wines«) in vier Versionen: Zwei linearen (Papier und Texteditor) sowie zwei nicht-

linearen (Hypertext-Softwareprodukte *HyperTIES* und *Hypercard*). Tests zu den behaupteten Fakten ergaben auch hier Präferenzen zugunsten der linearen Textpräsentation.

- Weitere Studien von Allinson/Hammond (1988/89) und Vickers/Gaines (1988) gehen der Frage nach, wie eine Hypertext-Basis beschaffen sein muss, um Lernen effektiv zu unterstützen.

### **Resümee: Hypertext und menschliches Verstehen**

Naive Überhöhung des Mediums Hypertext ist sicher unbegründet. Zu allen Argumenten für das neue Medium lassen sich Gegenargumente finden. Empirische Studien ergeben ein uneinheitliches Bild: zum Teil zugunsten von Hypertext, zum Teil zugunsten linearer Wissensdarbietung.

Am erfolgreichsten wird Hypertext derzeit in der technischen Dokumentation eingesetzt. Das ist nicht weiter verwunderlich, denn technische Dokumentation ist eine Sorte Literatur, die kaum jemand gerne freiwillig von vorne bis hinten liest. Auch in der HTML-Version dieses Buchs, werden Sie Kapitel finden, die Sie sicherlich nicht auf die gleiche Weise lesen werden wie Sie es hier in der Einführung vielleicht tun. Hypertext spielt immer dann seine Stärken aus, wenn sich der darzustellende Inhalt sinnvoll fragmentieren lässt. Ein gewaltsames Fragmentieren jeglicher Inhalte nach dem Motto »aber es soll doch Hypertext sein!« wird dagegen sein Ziel verfehlen. HTML, das Basisformat für Web-Seiten, zwingt Sie nicht zu gewaltsamer Fragmentierung, erlaubt sie jedoch. Insofern sind Sie bei der Informationsverteilung auf Web-Seiten frei und können selbst entscheiden, in welchem Ausmaß Sie Inhalte fragmentieren. Die Art der jeweiligen Inhalte sollte dabei das Maß sein. Wenn sich Inhalte gut fragmentieren lassen, dann sollte das Web-Design solcher Inhalte die Möglichkeiten von Fragmentierung und Vernetzung auch nutzen. Bei Inhalten, die sich nicht sinnvoll fragmentieren lassen, darf ein Text dagegen ruhig ein zusammenhängender Text bleiben.

## **1.6 Hilfsmittel**

### **1.6.1 Dokumente und Dokumentationen**

Die Fülle der Techniken und Technologien, die im Web und im Internet allgemein kursieren und eingesetzt werden, ist gewaltig. Kein Mensch ist in der Lage, alles, was dazugehört, zu kennen und zu beherrschen. Es ist deshalb essentiell wichtig, dass alle eingesetzten Techniken und Technologien möglichst gut und vollständig dokumentiert sind, und dass diese Dokumentationen offen zugänglich und einsehbar sind. Technische Dokumentation ist im Internet das Salz der Erde. Ohne Dokumentation gäbe es kein Internet – denn die Techniken und Technologien allein würden an die Köpfe derer gebunden bleiben, die sie beherrschen, sich nicht verbreiten und schnell in Vergessenheit geraten. Dokumentation ist also die Schriftkultur des Internet und die Basis für die Stabilität des Internet. Die meisten Menschen, die sich wirklich tiefer auf das Internet einlassen, begreifen das auch früher oder später. In den Jahrzehnten, die das Netz nun schon an Erfahrung gewinnen konnte, haben sich gewisse Standards beim Dokumentieren etabliert. Das betrifft sowohl die Art und Weise der Beschreibungen, als auch den Verbindlichkeits-Charakter.

Als Entwickler von Web-Seiten werden Sie kaum mit einer Dokumentation wie der vorliegenden alleine auskommen, auch wenn diese noch so umfangreich und ausführlich erscheinen mag. Denn es gibt einerseits Dokumente, die spezielle Themen einfach vollständig behandeln, was im

Rahmen einer übergreifenden Dokumentation wie der vorliegenden niemals möglich ist. Und andererseits gibt es Dokumente, die einen höheren Verbindlichkeits-Charakter haben als die Dokumentation, die Sie hier lesen. Denn nicht, was hier steht, ist beispielsweise der Maßstab dafür, wie HTML auszusehen hat, sondern das, was in dem dafür vorgesehenen offiziellen Dokument des W3-Konsortiums steht. Die verschiedenen Dokumentationsformen haben alle ihren Zweck und ihre Daseinsberechtigung. Aber Sie sollten die Unterschiede kennen und wissen, welche Dokumentationsform welche Aufgaben hat. Und schließlich müssen Sie natürlich auch wissen, wie Sie an die entsprechenden Dokumente herankommen.

Praktisch alle wichtigen Dokumente im Internet liegen in englischer Sprache vor. Viele davon sind auch nicht in Übersetzungen verfügbar, und wenn, dann gelten die englischsprachigen Versionen als die normative Grundlage. Der Einsatz automatischer Übersetzungshilfen führt bei solchen Texten in der Regel zu erheiternden Momenten, hilft aber beim Verständnis der Texte kaum weiter. Gute Kenntnisse der englischen Sprache sind also eine wichtige Voraussetzung, um die Originaldokumente lesen zu können.

### **Recommendations des W3-Konsortiums**

Das W3-Konsortium ist ein unabhängiges, sich selber überwachendes Gremium für die technischen Standards im Web. Recommendations (*Empfehlungen*) des W3-Konsortiums sind Dokumente mit dem höchsten im Web verfügbaren Verbindlichkeits-Charakter. Es handelt sich um technische Beschreibungen einzelner Web-Technologien wie beispielsweise HTML, CSS oder XML. Die Recommendations haben die Aufgabe, die jeweilige Technologie vollständig und unmissverständlich darzustellen. Die Zielgruppen dieser Beschreibungen sind einerseits Web-Autoren, aber auch Software-Hersteller, z.B. von Web-Browsern, die angehalten sind, diese Standards so genau und vollständig wie möglich in ihren Produkten zu implementieren. Der Anspruch der Exaktheit und Vollständigkeit macht die Recommendations für Anfänger nicht unbedingt zu einer leichten Lektüre. Denn nicht selten leidet die Allgemeinverständlichkeit unter den Zwängen der Terminologie. Das W3-Konsortium ist sich aber bewusst, dass diese Dokumente zum Teil auch von weniger spezialisierten Leuten und Nicht-Informatikern gelesen werden oder gelesen werden müssen. Die neueren Recommendations sind daher reichhaltiger erläuternd als die früheren. Auch mit erhellenden Beispielen wird nicht mehr so gespart.

Jede Recommendation durchläuft ein längeres Verfahren, in dessen Verlauf das Dokument mehrmals von verschiedenen Seiten gereviewt wird. Aus losen technischen Notizen entstehen zunächst **Working Drafts** (*Arbeitsentwürfe*), von denen es eine oder mehrere geben kann. Arbeitsentwürfe lassen das geplante End-Dokument bereits gut erkennen. Verbindlich sind sie jedoch noch nicht. Nach diversen Reviews innerhalb der Arbeitsgruppe und nach Abstimmung mit der Interessengruppe erlangt das Dokument schließlich den Status eines **Candidate Recommendation** (*Empfehlungskandidat*). In diesem Stadium wird noch Feedback aus der Öffentlichkeit erwartet und gegebenenfalls eingearbeitet. Daraus entsteht die **Proposed Recommendation** (*Empfehlungsvorschlag*). Nach einem abschließenden Review erhält das Dokument schließlich den Status einer offiziellen **Recommendation**.

Die Entwicklung der jeweiligen Technologie ist damit natürlich nicht abgeschlossen. Es handelt sich lediglich um festgeschriebene Entwicklungsstadien, damit Entwickler Fixpunkte haben, an die sie sich halten können. Die Recommendations erhalten aus diesem Grund Versionsnummern, ähnlich wie bei Software-Produkten üblich. So gibt es beispielsweise HTML 4.01, CSS 2.0 oder XML 1.0. Für neue Dokumentversionen wird das Review-Verfahren ganz oder teilweise wiederholt. Sämt-

liche Recommendations des W3-Konsortiums, auch in ihren vorausgehenden Review-Stufen, sind über folgende Web-Adresse erreichbar:

- [www.w3.org/TR/](http://www.w3.org/TR/)  
Das »TR« steht dabei für **Technical Reports** (*technische Berichte*). Sämtliche Dokumente werden in englischer Sprache veröffentlicht, und die englischsprachige Version ist die normative Version. Viele der Dokumente sind aber auch bereits in andere Sprachen übersetzt worden, auch ins Deutsche. Über die folgende Adresse können Sie verfügbare Übersetzungen aufrufen:
- [www.w3.org/Consortium/Translation/](http://www.w3.org/Consortium/Translation/)  
Alle wichtigen Recommendations des W3-Konsortiums liegen im HTML-Format vor und können von den Web-Seiten des W3-Konsortiums in gezippter Form downgeloadet werden. Auf diese Weise kann sich jeder Web-Entwickler lokale Kopien derjenigen Dokumente besorgen, die er für seine tägliche Arbeit benötigt. Wenn Sie sich ernsthaft mit HTML, CSS, XML usw. beschäftigen wollen, ist es dringend zu empfehlen, sich diese Dokumente zu besorgen und sich damit zu beschäftigen.

### **Requests For Comments (RFCs)**

Während die Recommendations des W3-Konsortiums speziell für das World Wide Web gelten, gibt es auch Dokumente für technische Standards im Internet allgemein. Das Verfahren für diese Standards ist nicht ganz so straff organisiert wie es durch die Organisation des W3-Konsortiums möglich ist. Doch das Verfahren hat sich bewährt, und die Dokumente gelten für Entwickler als verbindliche Grundlagen.

Die **Requests for Comments (RFCs)** (wörtlich: »Antrag auf Kommentierung«) sind der Dokumenttyp für solche Beschreibungen. Die Sammlung der RFCs wird seit den Anfangstagen des ARPANET, also seit 1969, geführt. Neue RFCs erhalten fortlaufende Nummern. So entstehen Dokumentnamen wie RFC 959 oder RFC 1867. Einer der RFCs, der RFC mit der Nummer 1543, beschreibt, wie RFCs auszusehen haben. Denn nur wenn ein einheitliches Schema eingehalten wird, ist sichergestellt, dass die RFCs eine gleichbleibende Qualität haben. Verwaltet werden die RFCs heute von der Organisation Internet Engineering Task Force (IETF, [www.ietf.org/](http://www.ietf.org/)).

Jeder Internet-Teilnehmer kann zu neuen RFCs beitragen. Wer einen neuen Standard vorschlagen möchte, z.B. ein bestimmtes Dateiformat oder ein bestimmtes technisches Ablaufverfahren, der kann einen so genannten Internet Draft (Internet-Entwurf) verfassen. Dabei hält er sich bereits an die in der RFC 1543 festgelegten Regeln, wie die Beschreibung auszusehen hat. Den fertigen Internet-Draft reicht er bei der IETF-Organisation ein. Dort gibt es eine Mailing-Liste, die alle an technischen Standards interessierten Internet-Teilnehmer abonnieren können. Neue Internet-Drafts werden in dieser Mailing-Liste bekannt gemacht und können diskutiert werden. Aus den Diskussionen ergeben sich in der Regel noch Korrekturen, Verbesserungsvorschläge usw., die der Verfasser der Internet Draft in sein Dokument einbringt. Nach einer angemessenen Diskussionsphase entscheidet schließlich die IETF-Organisation darüber, ob das Dokument ein offizieller RFC wird. Wenn ja, erhält es eine RFC-Nummer und wird in die Sammlung der RFCs aufgenommen.

RFCs enthalten viel Wissen, mitunter aber auch persönliche Ansichten der Autoren oder humorvolle Passagen. Viele wichtige defacto-Standards im Internet, etwa das HTTP-Protokoll oder die MIME-Technik für E-Mails, liegen in Gestalt einer oder mehrerer RFCs vor. Aus RFCs sind auch schon häufig Teile von W3C-Recommendations hervorgegangen. So liegen etwa den heute im

HTML-Standard verankerten Elementen für Multimedia-Objekte oder Tabellen ursprünglich RFCs zugrunde.

RFCs sind im Original reine Textdateien und durchgängig in englischer Sprache verfasst. Über die Web-Adresse *rfc.net* können Sie die aktuelle Sammlung der RFCs aufrufen. Im deutschen Internetraum werden die RFCs ebenfalls angeboten, z.B. von der FH Köln. Diese bietet die RFCs im HTML Format an (*rfc.fh-koeln.de/rfc.html*). Es ist sehr empfehlenswert, sich die Liste der RFCs mal in Ruhe anzusehen und einzelne RFCs daraus aufzurufen. Als Web-Entwickler sollten Sie den Dokumenttyp der RFCs kennen und wissen, dass in dieser Sammlung etliche Standards beschrieben sind, die Sie möglicherweise mal brauchen.

### **Frequently Asked Questions (FAQs)**

Frequently asked questions (*häufig gestellte Fragen*) sind keine offiziellen Dokumente in dem Sinn, dass sie von einer Organisation herausgegeben und verwaltet werden. Es handelt sich einfach um einen Dokumentationstyp, der sich im Internet etabliert hat und eine lange Tradition hat. Es gibt FAQs zu allen nur erdenklichen Themen – von Tieren über Politik und Religion bis hin zu technischen Fragen zum Internet. Im Web gibt es unter der Domain *faqs.org* eine Sammelstelle für FAQs. Es handelt sich dabei um eine Sammlung mit Verweisen zu FAQs.

FAQs basieren auf Fragen, die in Mailing-Listen, Newsgroups oder Foren – eben überall dort, wo Internet-Teilnehmer sich gegenseitig austauschen können – immer wieder auftauchen und gestellt werden. Eine beliebte Frage in Foren für Web-Seiten-Entwickler lautet beispielsweise: »wie programmiere ich einen Passwortschutz für meine Seiten?«. Anstatt eine Dokumentation zu schreiben, die sich inhaltlich in Kapitel, Themen, Unterthemen usw. aufgliedert, enthält eine FAQ als Unterüberschriften genau diese Fragen, als Frage formuliert. Unterhalb davon wird dann eine ordentliche und zielführende Antwort gegeben. In den Diskussionsbereichen, wo solche Fragen immer wieder auftauchen, können die »alten Hasen« dann, bevor sie nervlich explodieren, weil die gleiche Frage schon zum dritten mal innerhalb von zwei Tagen gestellt wurde, einfach auf die FAQ verweisen. FAQs sind deshalb mittlerweile auch meistens im Web in Form von Web-Seiten verfügbar – denn das ist die bequemste Form, den Fragesteller direkt auf die Antwort zu seiner Frage zu verweisen.

Gerade wenn Sie im Netz auf der Suche nach Support sind und sich noch nicht sonderlich auskennen, sollten Sie auf jeden Fall folgsam sein, wenn man Sie auf eine FAQ verweist. Denn FAQs sind aus der Praxis entstanden und tragen dazu bei, dass sich Hilfestellungen auf Probleme konzentrieren können, die nicht so alltäglich sind.

### **Anbieter-Dokumentationen**

Alle Web-Technologien, die nicht vom W3-Konsortium standardisiert werden, sind trotzdem irgendwo »im Original« dokumentiert. Denn hinter solchen Technologien, sei es nun Java, Perl, PHP usw., stecken Anbieter, die ein Interesse an der Verbreitung ihrer Technologie haben. Verbreiten kann sich eine Technologie aber nur, wenn sie ordentlich dokumentiert ist. Deshalb finden Sie normalerweise auf den Web-Seiten der Technologie-Anbieter die verbindlichen Dokumentationen für die jeweilige Technologie. Wichtig sind für Web-Entwickler auch die Entwickler-Seiten der großen Browser-Anbieter. Die beiden folgenden Angebote sollte jeder, der sich mit dem Erstellen von Web-Seiten beschäftigt, kennen:



- Entwickler-Seiten von Netscape (*developer.netscape.com/*)  
Dort finden Sie beispielsweise die Original-Dokumentationen zu JavaScript, aber auch Informationen, welche HTML-Elemente von welcher Netscape-Version unterstützt werden. Zahlreiche technische Notizen und Artikel verraten bestimmte Tricks und Kniffe, wie man so programmiert, dass der Netscape-Browser es korrekt interpretiert.
- Entwickler-Seiten von Microsoft (*msdn.microsoft.com/*)  
Dort finden Sie beispielsweise die Original-Dokumentationen zu JScript, der Microsoft-Implementierung von JavaScript. Ein wenig suchen müssen Sie allerdings, denn dort werden auch andere Technologien beschrieben, die nichts mit Web-Entwicklung zu tun haben. Der Informationsbestand ist gewaltig.

### **Freie Dokumentationen**

Freie Dokumentationen sind solche wie die, die Sie gerade lesen. Solche Dokumentationen können von jedem Autor, der sich dazu bemüht, im Netz veröffentlicht werden. Es gibt jedoch keinerlei kontrollierende Instanz dafür, und deshalb gibt es auch keine Garantie dafür, wie vollständig und sachlich richtig solche Beschreibungen sind. In der Regel entscheidet allerdings die Netzgemeinde über den Wert solcher Dokumentationen. Gute Dokumentationen werden von vielen anderen Web-Seiten aus verlinkt, werden dadurch immer bekannter und erreichen auf diese Weise den Status einer anerkannt brauchbaren Informationsquelle.

Eine wichtige Aufgabe der freien Dokumentationen ist es vor allem, das Wissen über Technologien in die einzelnen Landessprachen zu tragen. Denn gerade Anfänger tun sich nun mal leichter damit, eine neue, komplizierte Materie erst mal in der eigenen Sprache vermittelt zu bekommen. Und Menschen, die des Englischen nicht oder kaum mächtig sind, haben gar keine andere Wahl, als das Wissen über vermittelnde Quellen der eigenen Sprache zu erwerben. Im Linkverzeichnis des Online-Angebots von SELFHTML bei *selfaktuell.teamone.de* aktuell finden Sie aktuelle Verweise auf freie Dokumentationen in folgenden Seiten.

### **Bücher**

Wenn Sie die EDV-Fachabteilung einer Großstadt-Buchhandlung betreten, können Sie so viele Bücher rund um Themen zum Entwickeln von Web-Seiten mit nach Hause nehmen, dass Sie mehrere Regalwände damit füllen können. Vorausgesetzt, der Geldbeutel lässt es zu, denn preiswert sind solche Fachbücher fast nie. Dafür sind viele dieser Bücher schlecht. Aber nicht alle, manche sind auch hervorragend. Der Grund, warum es so viele schlechte Computer-Fachbücher gibt, liegt darin, dass es einerseits einen riesigen Lesemarkt für diese Fachliteratur gibt, aber nicht genügend Autoren mit echtem »Tiefenwissen«. Es gibt zwar jede Menge gute Programmierer und Fachleute mit dem nötigen Wissen, doch die sind meistens mit Projekten ausgelastet und auch oft nicht in der Lage, ein Buch zu schreiben. Also suchen die EDV-Verlage, den großen Absatzmarkt im Auge, oft händerringend nach Autoren und gehen dabei nicht selten das Risiko ein, einen Titel zu produzieren, bei dem die Qualität zu kurz kommt. Meistens haben die Autoren auch nicht mehr als zwei, drei Monate Zeit, um Bücher von mehreren hundert Seiten aus dem Boden zu stampfen. Das verleitet natürlich dazu, woanders abzukupfern, ohne den Stoff wirklich zu durchdringen, Inhalte mit »Füllmaterial« zu strecken, uneinheitliche Terminologie zu verwenden usw. – vor allem dann, wenn der Autor nicht aus einem vorhandenen Praxiswissen schöpfen kann, sondern sich während der Erstellungsphase die Materie selber erst erarbeiten

muss. Viele Bücher werden auch für den »schnellen Markt« produziert, d.h. sie werden ein paar Monate vom Verlag lang stark beworben, und dann verschwinden sie wieder, weil die Kalkulation des Verlags aufgegangen ist und ein weiteres Bewerben zu aufwendig wäre.

Bevor Sie sich Fachbücher anschaffen, sollten Sie daher nach Rezensionen suchen. Ganz hervorragend dazu geeignet sind die großen Internet-Buchanbieter wie Amazon ([www.amazon.de/](http://www.amazon.de/)). Auch wenn Sie lieber in einer realen Buchhandlung kaufen, lohnt es sich, dort Vorab-Informationen einzuholen. Denn Leser können dort zu Büchern, die sie gelesen haben, Rezensionen schreiben. Bei vielen Büchern kommen auf diese Weise mehrere Rezensionen zusammen, die sehr authentisch sind, weil sie unmittelbare Erfahrungen von Lesern widerspiegeln. Suchen Sie einfach nach Büchern zu einem gewünschten Thema, z.B. zu Büchern über HTML oder Perl, und lesen Sie, was andere darüber zu berichten wissen. Das ist zumindest sinnvoller, als sich nur von den Klappentexten der Bücher leiten zu lassen.

Generell lässt sich sagen, dass Fachbücher um so besser werden, je ausgereifter eine Technologie ist. Wenn neue, von der Fachszene als bedeutsam eingeschätzte Technologien entstehen, wie es in den letzten Jahren beispielsweise bei Java oder XML der Fall war, dann überstürzen sich viele Verlage geradezu, um bei den ersten zu sein, die etwas zu dem betreffenden Thema publizieren. Was dabei herauskommt, ist aber fast immer Mangelware, da die Technologien meist selber noch nicht ordentlich dokumentiert und noch in Entwicklung begriffen sind. In solchen Phasen ist es allemal besser, sich an kostenlose Dokumente im Internet zu halten. Ist eine Technologie dagegen gut eingeführt und etabliert, wie es mittlerweile etwa bei HTML der Fall ist, dann gibt es in der Regel auch ausgereifte Fachbücher dazu.

## 1.6.2 Software

### ***Allgemeines zu Software für Web-Entwickler***

Das wichtigste, was es zum Thema Software in Bezug auf das Entwickeln von Web-Seiten zu sagen gibt, ist, dass es »die Software« nicht gibt und hoffentlich auch nie geben wird. Lediglich bei proprietären Technologien wie Flash sind Sie auf bestimmte Software-Produkte angewiesen. Alle internet-genuinen Technologien sind dagegen bewusst software-unabhängig ausgelegt. Das hat gute Gründe. Im Internet treffen alle nur erdenklichen Betriebssysteme und Rechnertypen aufeinander, und Technologien, die für alle verfügbar sein sollen, haben keine große Chancen, wenn sie systemspezifisch sind oder spezielle Anforderungen stellen, die Teile der Internet-Nutzer ausschließt.

Die Software-Unabhängigkeit der Technologien führt auch dazu, dass Software-Hersteller keine Kundenbindung mehr über Dateiformate erzwingen können und dem Kunden dafür unflexible und unpraktische Anwendungsoberflächen zumuten. Software-Produkte können sich bei software-unabhängigen Daten nur dadurch gegenüber der Konkurrenz profilieren, indem sie den Anwender durch mehr Effizienz, bessere Übersichtlichkeit und eine intelligentere Oberfläche überzeugen. Software-Hersteller werden dadurch gezwungen, sich mehr Gedanken über Anwender-Ergonomie und intelligentes Oberflächen-Design zu machen.

Immer wieder treten Software-Produkte allerdings mit dem Anspruch an, dem Anwender ein Rundum-Sorglos-Paket zu anzubieten, das ihm das Erstellen, Verwalten und Überwachen der eigenen Web-Seiten mit dem Zusatz »kinderleicht und ohne tiefere Kenntnisse bedienbar« anpreist. Das stimmt aber meistens nicht. Gerade solche Software-Pakete sind meist hoffnungs-

los mit Funktionen, Menüs, Assistenten und Dialogen überladen. Der Anwender verbringt dann erst mal viel Zeit damit, sich den Umgang mit der Software anzueignen. Diese Zeit könnte in vielen Fällen sinnvoller darin investiert werden, sich direkt mit den dahintersteckenden Technologien zu beschäftigen.

Genauso dumm ist jedoch die Behauptung, sauber arbeiten könne man nur mit archaischen Werkzeugen wie dem Windows-Notepad-Editor oder der FTP-Kommandozeile. Software ist dazu da, um Aufgaben zu erleichtern, und wo sie das tut, ist sie auch angebracht. Es gibt eine Menge ausgereifter Tools, die beim Erstellen von Web-Seiten Zeit sparen und Fehler vermeiden helfen. Es ist jedoch sinnlos, bestimmte Produkte zu empfehlen. Denn der persönliche Arbeitsstil und die Einschätzung des »Look and Feel« einer Software sind von Mensch zu Mensch verschieden. Verschiedene Produkte einmal auszuprobieren und zu versuchen, damit warm zu werden, bleibt Ihnen nicht erspart. Glücklicherweise sind die meisten Produkte, auch wenn sie kostenpflichtig sind, als Ausprobierversion im Internet downloadbar. So können Sie ein Produkt unverbindlich testen, bevor Sie sich entschließen, es zu erwerben. Von dieser Möglichkeit sollten Sie unbedingt Gebrauch machen.

Im Linkverzeichnis des Online-Angebots von SELFHTML aktuell bei [selffaktuell.teamone.de](http://selffaktuell.teamone.de) oder auf der Buch-CD finden Sie aktuelle Verweise zu Software-Anbietern.

### **HTML-Editoren**

Grundsätzlich kann man bei HTML-Editoren zwischen quelltextbasierten Editoren und WYSIWYG-Editoren unterscheiden (WYSIWYG = *What You See Is What You Get* – *Was Sie sehen ist das was rauskommt*). Bei quelltextbasierten Editoren arbeiten Sie direkt mit den HTML-Auszeichnungen. Solche Editoren verfügen in der Regel über eine Werkzeugleiste und diverse Menübefehle oder Dialoge, um die HTML-Auszeichnungen in den Text einzufügen. Die HTML-Auszeichnungen erscheinen sichtbar im Text, in der Regel andersfarbig und optisch gut erkennbar dargestellt (»Syntax-Highlighting«). WYSIWYG-Editoren bieten ebenfalls Werkzeugleisten, Menübefehle und Dialoge an, um das Setzen der HTML-Auszeichnungen zu unterstützen. Der Unterschied ist jedoch, dass die Auszeichnungen nicht angezeigt werden (bzw. nur auf Wunsch). Stattdessen wird der Text bereits so am Bildschirm angezeigt wie später im Web-Browser. Das bedeutet aber umgekehrt nicht unbedingt, dass der Text nachher in allen Web-Browsern genauso erscheint wie im Fenster des Wysiwyg-Editors – »Wysiwyg« ist eigentlich eine gefährliche Irreführung und ein Versprechen, das nur schwer einhaltbar ist angesichts der Browser-Vielfalt und der vielen Tücken in den Browsern.

Lassen Sie sich nicht zu sehr von dem häufig anzutreffenden Glaubenskrieg zwischen Anhängern und Gegnern der WYSIWYG-Editoren beeindrucken. Versuchen Sie herauszufinden, womit Sie persönlich am besten arbeiten können.

### **Web-Browser**

Web-Browser sind Visualisierungsprogramme für Web-Seiten. Web-Browser beherrschen das HTTP-Protokoll und können mit Web-Servern kommunizieren, um etwa die Daten einer Web-Adresse anzufordern. Alle Web-Browser beherrschen mehr oder weniger gut HTML, die meisten mittlerweile auch die ergänzenden Technologien CSS und JavaScript, allerdings uneinheitlich vollständig. Server-seitig dynamische Anwendungen wie CGI-Scripts oder PHP-Anweisungen sind für Browser unproblematisch, solange sie HTML-Code für den Browser erzeugen. Große

Unterschiede gibt es dagegen noch bei der Unterstützung von Multimedia. Mit Web-Browsern können Sie auch offline HTML-Dateien auf Ihrem Rechner öffnen. Das ist z.B. wichtig, wenn Sie Web-Seiten lokal und ohne Internetverbindung austesten wollen.

Als Entwickler von Web-Seiten sollten Sie nicht nur einen Browser auf Ihrem Rechner installiert haben, sondern so viele Produkte wie nur möglich. Mit welchem Browser Sie persönlich am liebsten surfen, bleibt Ihnen überlassen. Aber die Besucher Ihrer Seiten verwenden nun mal unterschiedliche Produkte, und deshalb sollten Sie Ihre Seiten stets mit mehreren Produkten überprüfen. Denn leider gibt es immer noch viele ärgerliche Unterschiede zwischen den Browsern im Detail. Und Ihr Ehrgeiz sollte sein, attraktive Web-Seiten anzubieten, die aber trotzdem keine Anwender ausschließen, nur weil diese nicht den gleichen Browser verwenden wie Sie selbst.

### ***XML-Werkzeuge***

Viele HTML-Editoren sind nicht nur für HTML geeignet, sondern auch gut als Texteditoren für andere Sprachen verwendbar. Dennoch gibt es spezielle XML-Editoren, da es beim Bearbeiten von XML-Daten häufig auf andere Dinge ankommt als beim Bearbeiten von HTML-Daten. Denn XML hat zunächst mal nichts mit Seiten-Layouts zu tun. Und XML-Daten müssen auch keine textorientierten Daten sein. Ebenso gut lassen sich mit XML auch grafische, akkustische oder formale Daten beschreiben. Dabei kommt es stärker darauf an, die Baumstruktur der Daten optimal dargestellt zu bekommen, und bei Verwendung von DTDs (Dokumenttyp-Definitionen, deren Regeln die eingegebenen Daten entsprechen müssen) ist es hilfreich, wenn der Editor gar keine Eingaben zulässt, die nicht den Regeln entsprechen.

### ***FTP-Programme***

Um die Dateien eines Web-Projekts ins Web auf den Server zu laden, wo Sie Speicherplatz zur Verfügung gestellt bekommen haben, brauchen Sie in der Regel Software, die das File-Transfer-Protokoll (FTP) unterstützt. Mit solchen Programmen können Sie ihre Projektdateien hochladen und auf dem Server-Rechner verwalten (umbenennen, löschen usw.). Die meisten modernen FTP-Programme unterscheiden sich äußerlich kaum von allseits bekannten Datei-Commandern oder Explorern. In einem Bereich zeigen sie die lokale Verzeichnisstruktur an, in einem anderen Bereich die Verzeichnisstruktur des aufgerufenen Server-Rechners. Durch Markieren und Ziehen mit der Maus lassen sich Dateien uploaden oder downloaden. Ebenso einfach lassen sich Dateien löschen oder umbenennen. FTP-Programme verfügen in der Regel auch über eine »Site«-Verwaltung. Dort können Sie für jede benutzte FTP-Adresse die Zugangsdaten und technische Einstellungen speichern.

### ***Grafikprogramme***

Grafikprogramme sind eine wichtige Ergänzung zum Erstellen aussagekräftiger Web-Seiten. Es gibt eine Reihe guter bis hervorragender Shareware-Programme zum Erstellen, Konvertieren und Nachbearbeiten von Grafiken. Wer professionelle grafische Ansprüche hat, wird allerdings kaum um ein teures Profi-Programm herumkommen, und sei es nur für Spezialaufgaben. Auf jeden Fall müssen die eingesetzten Programme die Dateiformate GIF und JPEG unterstützen, denn das sind die verbreiteten Grafikformate im Web, die praktisch alle Browser direkt anzeigen können. Zum Entstehungsprozess einer Grafik tragen jedoch auch häufig auch andere Programme bei, etwa vektor-orientierte Programme oder Raytracing-Programme.

### **Diverse Tools**

Es gibt eine Menge anderer Programme und Software-Typen, die Web-Entwickler sinnvoll einsetzen können. Dazu gehören auf jeden Fall Standard-Tools wie Pack- und Entpackprogramme für Dateiformate wie ZIP, RAR usw. Denn oft werden Sie beispielsweise Daten Ihres Web-Projekts per E-Mail übertragen, sei es an Kunden, Freunde usw. Um dabei die Datenmengen klein zu halten und statt viele Dateien nur eine an die Mail anhängen zu müssen, sind Packer-Programme wichtig.

Weiterhin sind oft andere Internet-Client-Programme wichtig. So können Sie bei vielen Providern auch mit Telnet/SSH auf Ihre Daten auf dem Server zugreifen. Im Linkverzeichnis des Online-Angebots von SELFHTML aktuell finden Sie aktuelle Verweise zu Programmen für Telnet und SSH:

### **1.6.3 Services im Web**

#### **Allgemeines zu Web-Services**

Was hier unter Web-Services verstanden wird, sind Web-Angebote, die anderen Web-Anbietern helfen, ihr eigenes Web-Angebot ins Web zu bringen oder um Features zu erweitern, die im Rahmen des eigenen Web-Angebots aufgrund fehlender Möglichkeiten nicht realisierbar sind. Solche Web-Services bieten beispielsweise fertige CGI-Scripts an, die Sie von Ihren eigenen Seiten aus aufrufen können. Auf diese Weise können Sie etwa Foren, Chats, Gästebücher oder Zugriffszähler in Ihre Web-Seiten integrieren, auch wenn Sie selber keine Möglichkeit haben, CGI-Scripts zu installieren, oder wenn Sie sich dem Selberprogrammieren nicht gewachsen fühlen. Einige dieser Web-Services sind werbefinanziert, d.h. es werden automatisch Werbe-Banner eingeblen-det, wenn Sie sie verwenden. Andere sind für eine geringe monatliche oder jährliche Gebühr nutzbar, und wieder andere sind völlig frei.

Im Linkverzeichnis des Online-Angebots von SELFHTML aktuell bei *selfaktuell.teamone.de* finden Sie Verweise zu Web-Seiten, die einen der folgenden Web-Services anbieten.

#### **Provider-Übersichten**

Die Providerfrage (*Provider = Anbieter*) steht normalerweise am Beginn jedes Vorhabens, eigene Seiten ins Web zu bringen. Falls Sie sich über die grundsätzlichen Möglichkeiten und Unterschiede informieren wollen, können Sie den Abschnitt *Web-Projekte publizieren* (Kapitel 25.2) durchlesen. Bereits für die Entscheidung, wie Sie Ihr Angebot hosten lassen wollen, wie viel Sie bereit sind dafür zu zahlen, und welche Leistungen Sie für welche Kosten erwarten können, gibt es Web-Services – nämlich Provider-Übersichten. Dort werden aktuelle Preis-/Leistungs-informationen zu etlichen Providern gesammelt und so bereitgestellt, dass Sie Preise und Leistungen direkt vergleichen können.

#### **CGI-Services für eigene dynamische Web-Seiten**

Viele Homepage-Anbieter wünschen sich zumindest ein Gästebuch, in dem sich Benutzer eintragen und ihren Kommentar zu den Seiten abgeben können. Die Kommentare sind dann in die Seiten selbst eingebunden und einsehbar. Dazu ist jedoch ein CGI-Script oder eine CGI ver-

gleichbare Technologie erforderlich. Ähnlich ist es, wenn Sie einen der mittlerweile allerdings nicht mehr so beliebten »Counter« auf Ihre Einstiegsseite einbinden wollen, also einen Zähler, der anzeigt, wie viele Besucher schon auf der Seite waren. Immer beliebter werden auch eigene, web-basierte Foren, wo die Besucher untereinander diskutieren können. Dazu noch einen Chat, und schon kann die Homepage als Kommunikationsplattform für Besucher dienen. Sinn macht so etwas natürlich erst, wenn Sie genügend Besucher auf Ihren Seiten haben. Eine weitere wichtige, häufig benötigte Anwendung ist ein so genannter Form-Mailer. So etwas können Sie gut gebrauchen, wenn Sie eigene Formulare auf Ihren Web-Seiten anbieten, beispielsweise, um von den Seitenbesuchern Feedback, Anfragen oder Vorbestellungen einzuholen.

All diese Features erfordern fortgeschrittene Web-Technologien wie CGI/Perl, PHP oder Java. Es gibt Web-Services, bei denen Sie sich selber und ohne weitere Programmierkenntnisse ein Gästebuch, ein Forum oder dergleichen einrichten können. Vom Anbieter erhalten Sie dann normalerweise Informationen, was Sie in Ihre eigenen Web-Seiten einbauen müssen, damit der Fremd-Service innerhalb Ihres eigenen Angebots funktioniert. In der Regel sind das bestimmte HTML-Quelltextbausteine, die Sie in Ihre HTML-Dateien an bestimmten Stellen einfügen müssen.

Einige dieser Dienste sind kostenlos, andere können Sie nur gegen Gebühr in Anspruch nehmen. Die Gebühren sind jedoch meist sehr gering (z.B. DM 20.-/Jahr für Zugriffszähler und Statistik für die eigene Homepage).

### ***Services zur Optimierung des eigenen Web-Angebots***

Diese Sorte von Web-Services hilft Ihnen beim Optimieren Ihrer Web-Seiten und bei der Fehlersuche. Der bekannteste Service dieser Art ist zweifellos der W3-Validator ([validator.w3.org/](http://validator.w3.org/)) des W3-Konsortiums, mit dessen Hilfe Sie online Ihre Web-Seiten auf fehlerfreie HTML-Syntax überprüfen können. Ähnliche Services gibt es auch für andere Sprachen. Ferner gibt es Services wie »Link-Checker«, wo Sie Ihre Seiten daraufhin überprüfen können, ob alle darin enthaltenen Verweise funktionieren, aber auch Services, bei denen Sie testen lassen können, wie gut oder schlecht Ihre Web-Seiten für Besucher erreichbar sind (die Internet-Anbindung kann von Provider zu Provider sehr unterschiedlich sein), oder wie lange die durchschnittlichen Ladezeiten Ihrer Seiten bei verschiedenen Internet-Zugangsverbindungen sind.

## **1.6.4 Support**

### ***Wo es Support gibt***

Wenn Sie sich stundenlang mit Problemen von der Art herumschlagen, dass Netscape eine Tabelle anders darstellt als der Internet Explorer, oder dass Sie bei einem JavaScript Fehlermeldungen erhalten, die Sie nicht verstehen, dann werden Sie sich wahrscheinlich »den entscheidenden Tipp« wünschen. Denn oft genügt ja nur ein Hinweis, um einen auf die richtige Fährte zu bringen und ein Problem zu lösen.

Mit solchen Problemen sind Sie allerdings nicht allein. Zigtausend andere haben gleiche, ähnliche oder wieder andere Probleme beim Erstellen ihrer Seiten. Nun gibt es aber gleichzeitig auch zahlreiche Menschen, die Gefallen daran finden, andere zu unterstützen. Diese Konstellation führt dazu, dass es im Netz sehr gut besuchte Orte gibt, an denen Erfahrungsaustausch über das Erstellen von Web-Seiten, über Programmierung usw. möglich ist.

Das größte und vermutlich auch kompetenteste Know-How-System für Fragen und Antworten im Internet ist wohl das **Usenet** mit seinen Newsgroups. In deutscher Sprache gibt es Newsgroups zu HTML/CSS, zu JavaScript, zu CGI/Perl, zu XML usw.

Die Teilnahme ist kostenlos, jedoch benötigen Sie einen News-Reader (unter Windows genügt beispielsweise das E-Mail-Programm Outlook Express, das zum erweiterten Lieferumfang des Internet Explorers gehört). Außerdem benötigen Sie einen News-Server, von dem Sie die Diskussionsgruppen empfangen können. Fragen Sie, wenn Sie keinen solchen News-Server kennen, Ihren Internet-Zugangsprovider danach, ob er einen solchen Server anbietet. Achten Sie, falls Sie ohnehin einen neuen Zugangs-Provider suchen, gegebenenfalls darauf, dass dieser auch einen Newsgroup-Server anbietet. Bei den meisten News-Servern müssen Sie sich mit Kennwort und Passwort einwählen. Beides erhalten Sie vom Anbieter des News-Servers, in der Regel also vom Zugangs-Provider.

Immer wichtiger werden neben den Newsgroups auch **web-basierte Diskussionsforen**. Es gibt sogar Web-Angebote, die nichts anderes anbieten als Foren zu bestimmten Themen. Andere wieder bieten Informationen zu diesen Themen an und haben ein angeschlossenes Forum.

Web-basierte Foren haben meist noch keine so lange Tradition wie die Newsgroups, und die Anbieter sind häufig unerfahren in der Rolle eines »virtuellen Kneipenwirts«. Die Qualität, das Flair und die Besucherfrequenz solcher Foren kann sehr unterschiedlich sein. Die meisten Foren sind kostenlos benutzbar, einige erfordern allerdings eine namentliche Anmeldung.

Falls Sie nicht so gerne öffentlich posten, gibt es auch noch die Möglichkeit, **Mailing-Listen** zu abonnieren, die sich mit einem Fachgebiet befassen. Um an einer Mailing-Liste teilzunehmen, genügt ein E-Mail-Programm. Sie abonnieren eine Mailing-Liste, indem Sie eine Mail mit einem bestimmten Inhalt an eine bestimmte Mail-Adresse senden – viele Mailing-Listen erlauben mittlerweile aber auch das Abonnieren via Web-Formular. Wenn Sie eine Mailing-Liste abonniert haben, erhalten Sie alle E-Mails, die an die Liste gesendet werden. Selber können Sie ebenfalls Mails an die Liste senden, die dann an alle anderen Listenteilnehmer versendet werden.

Im Linkverzeichnis des Online-Angebots von SELFHTML aktuell bei [selfaktuell.teamone.de](http://selfaktuell.teamone.de) finden Sie Verweise zu einschlägigen Newsgroups, web-basierten Foren und Mailing-Listen, die sich mit Themen rund um das Erstellen von Webseiten befassen.

Alle diese Diskussionsorte im Netz sind vorwiegend für Probleme mit software-unabhängigen Technologien wie HTML, CSS, XML usw. oder für Probleme mit freeware-basierten Technologien wie Perl, PHP usw. gedacht. Wenn Sie Software käuflich erworben haben und Probleme mit dem Produkt haben, ist immer noch der Produktanbieter die erste Anlaufstelle für Fragen. So unterhalten beispielsweise große Software-Anbieter wie Microsoft, Macromedia oder Adobe neben den üblichen Call-Centern eigene Diskussionsforen auf ihren Web-Seiten oder in Form von Newsgroups.

### **Wie freiwillige Hilfeleistung funktioniert**

Freiwillige Hilfe, wie sie an den weiter oben genannten Diskussionsorten geleistet wird, basiert auf Geben und Nehmen. Wenn Sie als Anfänger an einem solchen Diskussionsort mehrfach Hilfe suchen und auch finden, dann ist es nur fair gegenüber den anderen Teilnehmern, wenn Sie später auch mal ein paar Fragen anderer beantworten, sofern Sie die Antwort wissen. Solche Diskussionsorte leben von der Bereitschaft aller Teilnehmer, sich gegenseitig zu helfen. Mit einer

Mentalität vom Typ »Hilfe, aber plötzlich bitte!« werden Sie dort schnell anecken, und an den meisten der genannten Orte werden die Dinge, die Sie dann zu hören bekommen, nicht gerade zimperlich formuliert. Bei offenen Standards und Technologien wie HTML sind Sie nun mal kein behüteter Verbraucher mit Rechten auf Rückgabe, Funktionsgarantie und Rundum-Service. Eine Newsgroup oder ein Diskussionsforum ist kein Geschäft, wo Sie einen Gerät gekauft haben. Wenn Sie allerdings den richtigen Ton treffen, erhalten Sie an solchen Diskussionsorten häufig wesentlich umfassendere und engagiertere Hilfe als dort, wo Sie einen vertraglichen Support-Anspruch haben. »Den richtigen Ton treffen« betrifft allerdings weniger das rhetorische Geschick. Sprachliche Höchstleistungen werden nicht erwartet, wenn Sie etwas fragen, wohl aber ein paar andere Dinge:

- Versuchen Sie zunächst alles was in Ihrer Macht steht, um ein Problem selber zu lösen. Für erfahrene Forenleser ist leicht erkennbar, ob Sie bereits selber nach Lösungen gesucht haben oder aus Bequemlichkeit fragen.
- Wählen Sie einen angemessenen Titel für Ihren Beitrag. Titel wie »JavaScript-Problem!« sind nicht gerade sehr erhellend in einem Forum, in dem es um nichts anderes als um JavaScript geht. Der Titel sollte den Problembereich erkennen lassen, aber auch nicht zu lang sein. Das Problem selber gehört im Text des Beitrags geschildert.
- Beschreiben Sie Ihr Problem so exakt wie möglich. Beschreiben Sie, mit welcher Testumgebung das Problem besteht, welche Fehlermeldungen Sie eventuell erhalten, und was Sie bereits ausprobiert haben, um das Problem zu lösen. Fügen Sie gegebenenfalls auch Ausschnitte aus Quelltexten bei, von denen Sie wissen, dass dort das Problem steckt – kopieren Sie aber andererseits nicht seitenweise Quelltext in einen Diskussionsbeitrag, damit verärgern Sie die Leser nur. Eine ideale Lösung ist, wenn Sie problematische Seiten an eine sonst unverlinkte Stelle ins Web hochladen und die Adresse dann in der Frage angeben.
- Bringen Sie ein wenig Freundlichkeit in die Diskussionsgruppe und etwas Respekt vor der Tatsache mit, dass Sie es mit anderen Menschen und nicht mit einer anonymen Antwortmaschine zu tun haben. Sparen Sie nicht an einer lockeren, aber netten Anrede wie »liebe Leute!« – an Diskussionsorten mit jüngerem Publikum reicht auch ein schlichtes »hi!«. Benutzen Sie Ihren echten Namen und geben Sie, falls Sie z.B. im Formular eines Beitrags zu einem Web-Forum dazu gezwungen werden, Ihre echte E-Mail-Adresse an. Das schafft Vertrauen und erhöht die Chancen auf brauchbare Antworten. An den meisten Diskussionsorten reden sich grundsätzlich alle mit Du an. Solche Dinge werden Sie allerdings selber schnell feststellen, wenn Sie ein wenig mitlesen – was Sie auf jeden Fall tun sollten, bevor Sie selber etwas beitragen.
- Lesen Sie, bevor Sie zum ersten mal etwas beitragen, auch die FAQs, die es zu den meisten Newsgroups, Foren und Mailing-Listen gibt. Dort steht nämlich genau, was in der jeweiligen Diskussionsgruppe erwünscht und unerwünscht ist, welche Quellen Sie kennen sollten, bevor Sie fragen usw. Die tatsächlich am häufigsten gestellten Fachfragen werden dort meist ebenfalls genannt und beantwortet.

Eigentlich sollten diese paar Grundregeln selbstverständlich sein, doch leider wird immer wieder und massenhaft gegen sie verstoßen. In Diskussionsforen und Newsgroups tauchen auch nicht selten sozial geschädigte, sonst kontaktarme Menschen auf, die zu Extremverhalten neigen, bewusst versuchen die Gruppe zu provozieren oder Gefallen daran finden, Diskussionen durch einsilbige und unpassende Bemerkungen zu zerstören. Dagegen hilft in der Regel, Ruhe zu bewahren und sich nicht provozieren zu lassen.



## 2 Web-Technologien

### 2.1 HTML

#### 2.1.1 HTML als »lingua franca« des Web

Unter einer »lingua franca« versteht man eine »Brot- und Butter-Sprache«, eine Sprache, die jeder kennt, jeder spricht, jeder leicht erlernen kann und jeder braucht. HTML ist eine solche Sprache. Vom Web-Gründer Tim Berners-Lee entwickelt, wurde HTML im Zuge des Web-Booms zum erfolgreichsten und verbreitetsten Dateiformat der Welt. Immer wieder rümpfen Entwickler, die gerne alles, was einfach sein könnte, durch tausend Filter und Übersetzer jagen, um es kompliziert zu machen und ihren technischen Durst zu stillen, über HTML die Nase – eben weil es so einfach ist. Aber erstens ist es bei genauerem Hinsehen gar nicht so einfach – fast kein namhaftes Web-Angebot, egal ob Suchmaschine, Online-Magazin oder Firmenauftritt, ist bis heute in der Lage, fehlerfreies und standardkonformes HTML auf seinen Web-Seiten zu realisieren. Und zweitens reicht HTML für die Mehrzahl der Inhalte, die heute im Web angeboten werden, vollkommen aus. Denn HTML ist eine Sprache zur Strukturierung von Texten, wobei aber auch die Möglichkeit besteht, Grafiken und multimediale Inhalte in Form einer Referenz einzubinden und in den Text zu integrieren.

Mit HTML können Sie Überschriften, Textabsätze, Listen und Tabellen erzeugen. Sie können anklickbare Verweise auf beliebige andere Web-Seiten oder Datenquellen im Internet erzeugen. Nicht-textuelle Inhalte können Sie wie bereits erwähnt referenzieren. Sie können Formulare in den Text integrieren. Und last but not least bietet HTML Schnittstellen für Erweiterungssprachen wie CSS Stylesheets oder JavaScript an, mit deren Hilfe Sie HTML-Elemente nach Wunsch gestalten und formatieren oder Interaktion mit dem Anwender realisieren können.

HTML ist also sehr durchdacht und in ein Gesamtkonzept weiterer Sprachen eingebunden. Wer behauptet, HTML sei nichts mehr für professionelles Web-Publishing, outet sich meistens nur als feuchtünger Technik-Junkie und behauptet letztlich nur, normale Texte seien nichts mehr für professionelles Web-Publishing. Und wer das behauptet, stellt sich selber ins Abseits der gewachsenen schriftlichen Kultur. Solange es Autoren gibt, die Texte schreiben und dazu die üblichen Mittel zur Textstrukturierung benötigen, wird es aus webtechnischer Sicht Leute geben, die HTML gut brauchen können.

Als »lingua franca« ist HTML allerdings mit diversen spezielleren Anforderungen schlichtweg überfordert. HTML allein kann weder Grafik-Designer befriedigen, die jeden Pixel am Bildschirm kontrollieren wollen, noch Daten-Designer, die aus der Welt der relationalen Datenbanken kommen und sich anwendungsspezifische Lösungen wünschen. Deshalb gibt es heute Style-Sprachen wie CSS, und es gibt Lösungen wie XML, um anwendungsspezifisches Daten-Design zu ermöglichen.

All das ändert aber nichts daran, dass HTML eine hervorragend geeignete, standardisierte und wegen der weiten Verbreitung der Web-Browser praktisch überall verfügbare Sprache für Text und Hypertext darstellt. Das W3-Konsortium ([www.w3c.org/](http://www.w3c.org/)), das für die Standardisierung von HTML zuständig ist, ist zwar bemüht, HTML von allen Sünden der Anfangsjahre zu reinigen und es als einfache, reine Textstrukturierungssprache zu etablieren. Doch in der Praxis dient HTML heute auch als Basis zum Erstellen von Web-Seiten-Layouts, und daran wird sich wohl auch so schnell nichts ändern.

### 2.1.2 HTML als softwareunabhängiges Klartextformat

HTML ist ein so genanntes Klartextformat. HTML-Dateien können Sie mit jedem beliebigen Texteditor bearbeiten, der Daten als reine Textdateien abspeichern kann. Es gibt also keine bestimmte Software, die man zum Erstellen von HTML-Dateien benötigt. Zwar gibt es längst mächtige Programme, die auf das Editieren von HTML spezialisiert sind, doch das ändert nichts an der entscheidenden Eigenschaft: HTML ist nicht an irgendein bestimmtes, kommerzielles Softwareprodukt gebunden. Diese wichtige, vielleicht sogar wichtigste Eigenschaft von HTML sollten Sie immer im Auge behalten, wenn man Ihnen erzählen will, dass Web Publishing nur mit bestimmten Softwareprodukten möglich sei.

Da HTML ein Klartextformat ist, lässt es sich auch hervorragend mit Hilfe von Programmen generieren. Von dieser Möglichkeit machen beispielsweise CGI-Scripts Gebrauch. Wenn Sie im Web zum Beispiel einen Suchdienst benutzen und nach einer Suchanfrage die Ergebnisse präsentiert bekommen, dann ist das, was Sie am Bildschirm sehen, HTML-Code, der von einem Script generiert wurde.

### 2.1.3 HTML als Auszeichnungssprache

HTML bedeutet HyperText Markup Language. Es handelt sich dabei um eine Sprache, die mit Hilfe von SGML (Standard Generalized Markup Language) definiert wird. SGML ist als ISO-Norm 8879 festgeschrieben. Mittlerweile gibt es einen Ableger von HTML namens XHTML. In der Sprachversion 1.0 ist XHTML eine Redefinition von HTML mit Hilfe von XML. XML hat nämlich ähnliche Aufgaben wie SGML.

HTML ist eine so genannte Auszeichnungssprache (Markup Language). Sie hat die Aufgabe, die logischen Bestandteile eines textorientierten Dokuments zu beschreiben. Als Auszeichnungssprache bietet HTML daher die Möglichkeit an, typische Elemente eines textorientierten Dokuments, wie Überschriften, Textabsätze, Listen, Tabellen oder Grafikreferenzen, als solche auszuzeichnen.

Das Auszeichnungsschema von HTML geht von einer hierarchischen Gliederung aus. HTML zeichnet Inhalte von Dokumenten aus. Dokumente haben globale Eigenschaften wie zum Beispiel Kopfdaten. Der eigentliche Inhalt besteht aus Elementen, zum Beispiel einer Überschrift 1. Ordnung, Textabsätzen, Tabellen und Grafiken. Einige dieser Elemente haben wiederum Unter-elemente. So enthält ein Textabsatz zum Beispiel eine als betont oder fett markierte Textstelle, eine Aufzählungsliste besteht aus einzelnen Listenpunkten, und eine Tabelle gliedert sich in einzelne Tabellenzellen.

Die meisten dieser Elemente haben einen fest definierbaren Erstreckungsraum. So geht eine Überschrift vom ersten bis zum letzten Zeichen, eine Aufzählungsliste vom ersten bis zum letzten Listenpunkt, oder eine Tabelle von der ersten bis zur letzten Zelle. Auszeichnungen markieren Anfang und Ende von Elementen. Um etwa eine Überschrift auszuzeichnen, lautet das Schema:

```
[Überschrift] Text der Überschrift [Ende Überschrift]
```

Bei einem Element, das wiederum Unterelemente besitzt, etwa einer Aufzählungsliste, lässt sich das gleiche Schema anwenden:

```
[Liste]
[Listenpunkt] Text des Listenpunkts [Ende Listenpunkt]
[Listenpunkt] Text des Listenpunkts [Ende Listenpunkt]
[Ende Liste]
```

Web-Browser, die HTML-Dateien am Bildschirm anzeigen, lösen die Auszeichnungsmarkierungen auf und stellen die Elemente dann in optisch gut erkennbarer Form am Bildschirm dar. Dabei ist die Bildschirmdarstellung aber nicht die einzige denkbare Ausgabeform. HTML kann beispielsweise genauso gut mit Hilfe synthetisch erzeugter Stimmen auf Audio-Systemen ausgegeben werden.

### 2.1.4 HTML für Hypertext

Eine der wichtigsten Eigenschaften von HTML ist die Möglichkeit, Verweise zu definieren. Verweise (»Hyperlinks«) können zu anderen Stellen im eigenen Projekt führen, aber auch zu beliebigen anderen Adressen im World Wide Web und sogar zu Internet-Adressen, die nicht Teil des Web sind.

Durch diese einfache Grundeigenschaft eröffnet HTML völlig neue Welten. Das Bewegen zwischen räumlich weit entfernten Rechnern wird bei modernen grafischen Web-Browsern auf einen Mausklick reduziert. In Ihren eigenen HTML-Dateien können Sie Verweise notieren und dadurch inhaltliche Verknüpfungen zwischen Ihren eigenen Inhalten und denen anderer Anbieter herstellen. Auf dieser Grundidee beruht letztlich das gesamte World Wide Web, und dieser Grundidee verdankt es seinen Namen.

Im Zeitalter der Kommerzialisierung des Internet sind natürlich auch die Verweise zu einem kommerziellen Gegenstand geworden. Anklickbare Werbe-Grafiken (»Banner«) auf häufig besuchten Seiten führen zu Anbietern, die für die Platzierung der Banner Miete bezahlen. Auch das sind Verweise. Glücklicherweise gibt es daneben aber weiterhin genügend »herkömmliche« Verweise im WWW, die einfach nur die Grundidee des Web verfolgen und zur weltweiten Vernetzung von Information beitragen wollen.

### 2.1.5 HTML-Versionen

Die erste Sprachversion von HTML gilt heute nicht mehr als erwähnenswert. Die Spezifikation zu dieser HTML-Version ist auf den Seiten des W3-Konsortiums, das für die Standardisierung von HTML verantwortlich ist, auch gar nicht mehr verfügbar.

- HTML 1.0 enthielt Auszeichnungen für Standardelemente wie Überschriften, Textabsätze, für Grafikreferenzen und natürlich für Verweise.
- HTML 2.0 wurde im November 1995 offizieller Sprachstandard. Die Spezifikation für HTML ist beim W3-Konsortium noch verfügbar ([www.w3.org/MarkUp/html-spec/html-pubtext.html](http://www.w3.org/MarkUp/html-spec/html-pubtext.html)). Die Version 2.0 von HTML gilt heute als der allerkleinste gemeinsame Nenner. Allerdings enthält HTML 2.0 durchaus Sprachelemente, die bis auf den heutigen Tag von den großen Browsern nicht oder nicht vollständig unterstützt werden, etwa das `<link>`-Tag.  
HTML 2.0 wurde allgemein als Enttäuschung empfunden, da gerade Netscape in seiner Entwicklung schon viel weiter war: so war HTML 2.0 Standard, als der Navigator in seiner überwältigenden Produktversion 2.0 auf den Markt kam. Darin waren bereits solche Dinge wie Frames (Mehrfenster-technik) und Multimedia-Referenzen implementiert, die meilenweit über den offiziellen HTML-Standard hinausgingen. Die meisten Ersteller von Web-Seiten berauschten sich damals an den neuen Möglichkeiten von Netscape und hatten wenig Verständnis für die Restriktionen von HTML 2.0.
- HTML 3.2 wurde nach langen Diskussionen am 14.1.1997 offizieller Sprachstandard. Die Spezifikation zu HTML 3.2 ist auf den Seiten des W3-Konsortiums verfügbar ([www.w3.org/TR/REC-html32.html](http://www.w3.org/TR/REC-html32.html)).  
Dass diese Version eine schwere Geburt war, lässt sich schon an der merkwürdigen Versionsnummer erahnen. Das W3-Konsortium, von den Erfolgen eines Netscape-Browsers unbeeindruckt, entwickelte zunächst eine geplante Version 3.0. Die Vorschläge zu HTML 3.0 enthielten zwar etliche interessante Bestandteile. Doch insgesamt liefen die Vorschläge völlig an den neuen Realitäten im Web vorbei. HTML 3.2 ist das Ergebnis einer völligen Umarbeitung der Vorschläge zu dem niemals offiziell gewordenen HTML 3.0. Endlich wurden zumindest Tabellen offizieller Bestandteil von HTML. Daneben aber auch diverse Elemente zur physischen Textauszeichnung. Aus letzterem Grund stellt HTML 3.2 aus heutiger Sicht ein mittleres Unglück für die Entwicklung von HTML dar. Denn in dieser Version wurde versucht, HTML als Sprache für optisches Design zu etablieren, was jedoch nicht dem Wesen der Sprache entsprach und auch nur halbherzig implementiert war. Viele Sprachbestandteile, die in HTML 3.2 eingeführt wurden, sind mittlerweile als *deprecated* (missbilligt) eingestuft und sollen künftig wieder aus dem HTML-Sprachstandard entfallen, weil sie durch andere, ergänzende Technologien wie CSS realisierbar sind.
- HTML 4.0 wurde erstmals am 18.2.1998 als Sprachstandard verabschiedet. Diese Sprachversion wurde jedoch mehrfach überarbeitet und liegt mittlerweile in einer Version 4.01 vor. Die jeweils aktuelle Version von HTML 4.x finden Sie auf den Seiten des W3-Konsortiums ([www.w3.org/TR/html4](http://www.w3.org/TR/html4)).  
HTML 4.0 stellt den Versuch dar, sich auf die Kernaufgaben von HTML zurückzubedenken. Das Arsenal der Sprachbestandteile dieser Version regelt vor allem die Einbindung von CSS Style Sheets und von Script-Sprachen wie JavaScript in HTML. Ein weiterer wichtiger Schwerpunkt ist das Thema Internationalisierung. HTML 4.0 setzt im Gegensatz zu seinen Vorläufern voll auf das Unicode-System und erlaubt damit, Texte in allen nur erdenklichen Sprachen der Welt HTML-gerecht zu notieren. Es gibt allerdings auch viel Verwirrendes in HTML 4.0. Dazu gehören vor allem die drei Sprachvarianten, die das W3-Konsortium anbietet: »HTML Strict«, »HTML Transitional« und »HTML Frameset«. Für Anfänger ist die Politik, die hinter dieser Aufteilung steckt, kaum nachvollziehbar, und so ist es kein Wunder, dass sich die meisten Web-Entwickler bis heute wenig Gedanken machen über diese Feinheiten. Das ist zwar schade, weil es sich durchaus lohnt, mit den Bestrebungen des W3-Konsortiums

eingehender auseinander zu setzen. Andererseits ist es verständlich, dass jemand, der »mal eben« seine Homepage im Web veröffentlichen will, keine Lust hat, sich mit konzepttheoretischen Aspekten des Designs von Auszeichnungssprachen zu beschäftigen. Mit diesem Widerspruch lebt HTML 4.0 leider.

Verkompliziert wurde die Sache außerdem noch dadurch, dass HTML im Zuge der Etablierung von XML in Gestalt von XHTML neu definiert wurde. So existiert heute neben dem SGML-basierten, klassischen HTML das XML-basierte XHTML. Letzteres ist zwar auch »nur« HTML, aber aufgrund XML-bedingter Gegebenheiten gibt es einige Unterschiede zu HTML. Von XHTML gibt es mittlerweile auch schon zwei Versionen: eine Version 1.0 ([www.w3.org/TR/xhtml1/](http://www.w3.org/TR/xhtml1/)) und eine Version 1.1 ([www.w3.org/TR/xhtml11/](http://www.w3.org/TR/xhtml11/)).

Angesichts dieser verwirrenden Versionen-Politik, die ein Kompromiss aus den Vorstellungen verschiedener Seiten und Interessen darstellt, gilt es für HTML-Anfänger vor allem, Ruhe zu bewahren und nicht gleich angewidert davonzurennen. Es ist durchaus möglich, HTML-4.0 oder XHTML-1.0-gerechte Web-Seiten zu erstellen, die richtig gut aussehen. Es gehört einfach ein wenig Kenntnis der Sachlage dazu. Innerhalb des HTML-Kapitels der vorliegenden Dokumentation wird versucht, die »Philosophie« des heutigen HTML so zu vermitteln, dass keine Entmutigungseffekte eintreten.

## 2.2 CSS Stylesheets

### 2.2.1 Formatsprache für HTML-Elemente

Mit HTML definieren Sie den grundsätzlichen Aufbau Ihrer Web-Seiten, die Elemente, die Strukturen, die Verweise, die referenzierten Elemente wie Grafiken, Multimedia usw. Aber HTML ist eigentlich nicht dazu gedacht, um exakt anzugeben, wie ein Element genau aussehen soll. So können Sie in HTML eine Überschrift definieren und dabei angeben, dass es sich um eine Überschrift 1. Ordnung handeln soll. Sie können innerhalb von HTML aber nicht bestimmen, wie groß, in welcher Schriftart usw. die Überschrift dargestellt werden soll. Das übernimmt der Web-Browser bei der Anzeige. Er benutzt dabei eine Mischung aus Grundeinstellungen, die der Anwender vornehmen kann, und fest programmierten Darstellungen einzelner HTML-Elemente.

An diesem Punkt setzen die Cascading Stylesheets (CSS) ein. Es handelt sich dabei um eine unmittelbare Ergänzungssprache, die vorwiegend für HTML entwickelt wurde. Sie klinkt sich nahtlos in HTML ein und erlaubt das beliebige Formatieren einzelner HTML-Elemente. Mit Hilfe von CSS Stylesheets können Sie beispielsweise festlegen, dass alle Überschriften 1. Ordnung 24 Punkt groß sind, in roter Helvetica-Schrift, mit einem Nachabstand von 16 Punkt und mit einer grünen doppelten Rahmenlinie oberhalb dargestellt werden. Sie können aber genauso gut auch für einen beliebigen Text festlegen, dass nur dieser Text 3 Zentimeter groß sein soll und eine gelbe Hintergrundfarbe erhält. Daneben enthält CSS auch Möglichkeiten zum punktgenauen Platzieren von Elementen am Bildschirm und für andere Ausgabemedien wie Drucker oder Audio-Systeme.

CSS erlaubt es, zentrale Formate zu definieren, beispielsweise für alle Überschriften erster Ordnung, oder für alle Textabsätze mit einem bestimmten Klassennamen, oder für kursiv ausge-

zeichneten Text, der innerhalb einer Tabellenzelle vorkommt. Die zentralen Formate können sich auf eine HTML-Datei beziehen, aber auch in eine externe Style-Datei ausgelagert werden, die Sie dann in allen gewünschten HTML-Dateien einbinden können. So werden einheitliche Formatvorgaben möglich, und der HTML-Code wird von unnötigem Ballast befreit.

CSS ist mit HTML eng verzahnt. Es lohnt sich daher, beide Sprachen gleichzeitig zu erlernen. Im HTML-Kapitel der vorliegenden Dokumentation werden Sie bei den Beschreibungen zu einzelnen HTML-Elementen deshalb immer auch auf die Möglichkeiten hingewiesen, wie Sie diese Elemente mit Hilfe von CSS gestalten können.

CSS ist ebenso wie HTML eine Klartextsprache. Auch für CSS brauchen Sie keine bestimmte Software, es genügt ein Texteditor. CSS ist wie HTML eine offen dokumentierte und vom W3-Konsortium standardisierte Sprache, die Sie frei und ohne Lizenzprobleme verwenden können.

Vom MS Internet Explorer wurde CSS schon recht früh unterstützt, da Microsoft sein Know-how aus dem Bereich der visuellen Textverarbeitung in den Browser einfließen lassen konnte. Netscape tat sich dagegen schwer mit der Implementierung. Was die immer noch verbreitete Version 4.x an CSS beherrscht, ist insgesamt wenig ermutigend, und Web-Designer müssen alle Augenblicke lang wieder mit bösen Überraschungen bei diesem Browser rechnen. Erst in der von Grund auf neu programmierten 6er-Version des Netscape-Browsers ist CSS sauber umgesetzt. Aber auch beim Internet Explorer gab es lange Zeit eigenwillige Interpretationen einzelner CSS-Eigenschaften. Auch hier kann man erst seit der 5er-Version eine ausgereifte Umsetzung erwarten.

Das Verhalten der Browser ist der Grund, warum CSS bis heute eher vorsichtig eingesetzt wird. Dennoch setzt mittlerweile die Mehrzahl der Angebote im Web CSS mit Erfolg ein, und solange Sie Ihre Seiten mit verschiedenen Browsern austesten – was Sie ohnehin tun sollten –, besteht kein Grund mehr, auf CSS und seine Vorteile zu verzichten.

### 2.2.2 CSS-Versionen

Das Standardisierungsverfahren bei CSS folgt den gleichen Regeln wie das von HTML. Die erste CSS-Sprachversion in Form einer offiziellen Empfehlung (*Recommendation*) stammt bereits vom 17.12.1996, wurde allerdings 1999 noch mal in überarbeiteter Form herausgegeben. Die Spezifikation zu CSS 1.0 ist auf den Seiten des W3-Konsortiums verfügbar ([www.w3.org/TR/REC-CSS1](http://www.w3.org/TR/REC-CSS1)).

Seitdem wurde CSS fleißig weiterentwickelt. Viele CSS-Eigenschaften, die längst beliebt geworden sind – etwa das absolute Positionieren von Elementen –, gehören zum »CSS Level 2«. Die Version 2.0 stammt vom 12.05.1998. Auch diese Version können Sie aufrufen ([www.w3.org/TR/REC-CSS2/](http://www.w3.org/TR/REC-CSS2/)).

Obwohl schon 1998 veröffentlicht, ist CSS 2.0 noch immer nicht vollständig in allen verbreiteten Browsern umgesetzt. Beim W3-Konsortium wird indessen längst an der Version 3.0 gearbeitet. Über die Einstiegsseite zu CSS ([www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)) beim W3-Konsortium können Sie sich über die laufende Weiterentwicklung informieren.

## 2.3 XML und XML-Derivate

### 2.3.1 XML als Definitionssprache für Auszeichnungssprachen

Stellen Sie sich vor, irgendjemand hätte irgendwann für ein weit verbreitetes Textverarbeitungsprogramm wie MS Word eine bestimmte Dokumentvorlage erstellt. Stellen Sie sich weiter vor, diese Dokumentvorlage würde sich irgendwie immer mehr verbreiten, jeder Word-Anwender hätte sie, und jeder Word-Anwender würde seine Dokumente fortan nur noch mit dieser einen Dokumentvorlage erstellen.

Genau das ist im Grunde bei HTML der Fall. HTML ist eine Sprache, die mit Hilfe von SGML definiert wird und eine Reihe von festen Elementen enthält. Es ist zwar eine Sprache, die eigentlich alle wichtigen Elemente enthält, die ein Web-Autor heute so braucht, doch es bleiben genügend Wünsche offen. Denn es gibt viele Texte, die anwendungsspezifische Datenstrukturen haben. Nehmen Sie beispielsweise E-Mails. Eine E-Mail hat bestimmte Kopfdaten wie Absender, Empfänger, Thema, Absendezeit sowie den eigentlichen Text. Wenn Sie eine solche E-Mail mit allen Feldern auf einer Web-Seite wiedergeben möchten, können Sie natürlich für die Felder »Absender«, »Empfänger« usw. einfach normale Textabsätze verwenden. Sinnvoller wäre es allerdings, Auszeichnungselemente mit Namen wie »Absender« oder »Empfänger« zu haben. Nur so würde die semantische Datenstruktur der E-Mail erhalten bleiben.

Und genau dies ist der Unterschied zwischen HTML und XML. XML wird auch mit Hilfe von SGML, der Mutter aller Auszeichnungssprachen, definiert. Doch es ist keine »SGML-Anwendung«, so wie HTML, sondern ein »SGML-Profil«. Das ist eine Stufe abstrakter, man spricht dabei auch von »generalized markup«, also von »verallgemeinerter Auszeichnung«.

Im Grunde ist XML auch gar keine Konkurrenz zu HTML, sondern vielmehr zu SGML. Denn mit Hilfe von XML können Sie wie mit SGML eigene, neue Sprachen »erfinden«. XML selbst ist also nur dazu da, um Auszeichnungssprachen zu definieren. Eine solche Auszeichnungssprache muss bestimmte Grundkonventionen einhalten. Die Bestandteile der Auszeichnungssprache müssen nach einem vorgeschriebenen Schema definiert werden. XML ist dabei das »Regelwerk«, das Schema. Beim Definieren eigener Auszeichnungssprachen mit XML können Sie eigene Namen vergeben. So können Sie zum Beispiel Elemente definieren, um Bereiche eines Dokuments als Lexikonartikel auszuzeichnen, als Musiknote oder als Bestandteil einer chemischen Formel. Sie müssen bei der Definition der Elemente jedoch exakt festlegen, welche Eigenschaften das Element hat, zum Beispiel, innerhalb welcher anderen Elemente es vorkommen kann und innerhalb welcher nicht. Wenn Sie dann ein XML-Dokument mit der Definition einer eigenen Sprache erstellt haben, können Sie Dateien in dieser Sprache erstellen und im Dateikopf angeben, auf welche XML-Definitionen Sie sich in dieser Datei beziehen.

Auszeichnungssprachen, die Sie mit XML definieren, sind wie HTML logisch/semantische Sprachen. Denn mit XML können Sie Elemente und deren grundsätzliche Eigenschaften definieren, aber Sie können nicht angeben, wie eine interpretierende Software ein Element bei der Darstellung formatieren soll. Da ein Browser bei selbst »erfundenen« Elementen nicht mal auf Voreinstellungen zurückgreifen kann, so wie bei HTML, ist eigentlich überhaupt keine sinnvolle Anzeige möglich. XML ist deshalb noch viel stärker als HTML auf die Ergänzung durch eine Style-Sprache angewiesen. Durch Formatierung von Elementen wird bei XML-Sprachen eine optisch brauchbare Darstellung überhaupt erst möglich.

Dazu gibt es, genau wie bei HTML, eine ergänzende Stylesheet-Sprache. Diese Sprache heißt **XSL** (*Extensible Stylesheet Language – erweiterbare Formatsprache*). XSL ist noch leistungsfähiger als CSS. Denn XSL besteht nicht nur aus Formateigenschaften wie Schriftgröße, Farbe, Zeilenabstand usw. XSL übernimmt auch erweiterte Aufgaben der Textverarbeitung wie das sortierte Ausgeben von listenartigen Daten, automatische Nummerierung usw.

Ein Ableger von XSL namens **XSLT** (*XSL Transformation*) erlaubt es sogar, Daten von einer XML-basierten Auszeichnungssprache in eine andere zu konvertieren. Das Konzept dazu ist ziemlich genial – XSLT können Sie sich vorstellen wie einen »Universal-Konverter«, einen Babelfish für Auszeichnungssprachen. Er kann von einer in die andere Sprache übersetzen. So ist es beispielsweise möglich, Daten in einer XML-basierten eigenen Sprache zu speichern, diese aber beim Präsentieren im Web auf dem Server, also bevor die Daten zum Browser gelangen, mit Hilfe von XSLT in HTML zu übersetzen. Beim Browser kommt dann lupenreines HTML an, und er bekommt gar nichts davon mit, dass die Daten eigentlich in einer für ihn möglicherweise unverständlichen XML-Sprache gespeichert sind.

Für normales Homepage-Design ist XML bis heute relativ uninteressant. HTML und CSS sind für normale Texte und deren Gestaltung am Bildschirm hervorragend geeignet und meistens ausreichend. Doch im professionellen Bereich gibt es genügend Fälle, in denen der Einsatz von XML seine Berechtigung hat. Eigentlich überall dort, wo anwendungsspezifische Datenstrukturen vorliegen und der Wunsch besteht, diese beim Speichern der Daten nicht aufzugeben (siehe E-Mail-Beispiel weiter oben).

Der Internet Explorer ist seit Version 5.x in der Lage, Dateien mit XML-basierten Sprachen zu erkennen und zu verarbeiten. Von XSL und XSLT versteht er allerdings noch recht wenig, gerade mal genug, um ein paar grobe Aufgaben der Darstellung damit zu lösen. Netscape erkennt XML seit der Produktversion 6.x. Angesichts dieser Tatsache ist es heute nicht sehr ratsam, XML direkt auf die Browser loszulassen. In der Praxis wird XML heute vor allem eingesetzt, um Daten semantisch optimal und frei von allem Layout-Ballast zu speichern. Für die Präsentation als Web-Seiten werden die Daten dann serverseitig mit Hilfe von XSLT oder mit Hilfe eines selbst geschriebenen CGI-Scripts in HTML übersetzt.

## 2.3.2 XML-Derivate

Auszeichnungssprachen, die mit Hilfe von XML definiert werden, werden auch als XML-Derivate, also als XML-Ableger bezeichnet. XML ist also vorstellbar als Stamm eines Baumes, der sich in lauter Äste entfaltet, wobei alle Äste die »Gene« des Stamms enthalten.

Neben Auszeichnungssprachen, die Sie mit XML für Ihre eigenen Zwecke definieren können, gibt es aber auch XML-basierte Sprachen, die für die allgemeine Verwendung vorgesehen sind. So bietet beispielsweise auch das W3-Konsortium gleich mehrere, mit Hilfe von XML definierte Auszeichnungssprachen öffentlich an in der Hoffnung, dass diese eine breitere Anwendung finden. Es handelt sich um Sprachen mit ganz unterschiedlichen Aufgaben. So gibt es beispielsweise ein Dateiformat namens **SVG**, mit dessen Hilfe sich Vektorgrafiken erzeugen lassen, oder eine Sprache namens **MathML** zum exakten Auszeichnen mathematisch-wissenschaftlicher Formeln. Wenn von XML die Rede ist, dann ist damit meistens nicht nur der Kern von XML gemeint, sondern auch eine ganze Familie von XML-basierten Sprachen – eben die XML-Derivate.



### 2.3.3 XML-Versionen

Genau wie HTML und CSS wird XML vom W3-Konsortium standardisiert. Die offizielle Dokumentation zu XML beschreibt dabei, nach welchen Regeln XML-basierte Auszeichnungssprachen definiert werden. Zum Redaktionsschluss des vorliegenden Dokuments lag die Empfehlung zu XML 1.0 vor, veröffentlicht am 10.02.1998. Auf den Seiten des W3-Konsortiums können Sie die jeweils aktuelle Version aufrufen ([www.w3.org/TR/REC-xml](http://www.w3.org/TR/REC-xml)). Auch XSL und XSLT werden vom W3-Konsortium standardisiert. Die entsprechenden aktuellen Dokumente finden Sie ebenfalls im Web:

- Aktuelle Spezifikation für XSL ([www.w3.org/TR/xsl/](http://www.w3.org/TR/xsl/))
- Aktuelle Spezifikation für XSL Transformations (XSLT, [www.w3.org/TR/xslt](http://www.w3.org/TR/xslt))

## 2.4 JavaScript/DOM

### 2.4.1 Web-Seiten als Anwendungen

In HTML können Sie unter anderem Formulare definieren. Solche Formulare können Eingabefelder, Auswahllisten, Buttons usw. enthalten. Der Anwender kann ein Formular ausfüllen und über das Web absenden. Das ist für viele Zwecke eine segensreiche Erfindung. Doch HTML erlaubt Ihnen als Formularanbieter beispielsweise nicht, die Eingaben des Anwenders vor dem Absenden des Formulars auf Vollständigkeit und Plausibilität zu prüfen. Mit JavaScript ist das jedoch möglich.

Oder ein anderer Fall: Sie können in HTML zwar Multimedia-Dateien einbinden, aber wenn diese HTML-Datei im Web steht, können Sie nicht wissen, ob Ihre Seitenbesucher einen Browser besitzen, der das entsprechende Multimedia-Format anzeigen kann. Da wäre es praktisch, das Einbinden der Multimedia-Datei davon abhängig zu machen, ob der Browser des Anwenders das entsprechende Format anzeigen kann oder nicht. Auch so etwas ist mit JavaScript möglich.

Für diese und unzählige andere nützliche, allerdings auch weniger nützliche Zwecke erfand Netscape eine Programmiersprache namens JavaScript. Durch JavaScript können Web-Seiten, die mit Hilfe von HTML aus strukturiertem Text bestehen und mit CSS gestaltet sind, ihren reinen »Dokument-Charakter« aufgeben und eher wie Programme, wie Anwendungen wirken. JavaScript erlaubt es, Maus- und Tastatureingaben des Anwenders zu verarbeiten und darauf mit Bildschirmausgaben oder dynamischen Änderungen innerhalb der angezeigten Web-Seite zu reagieren. So lässt sich mit JavaScript beispielsweise ein HTML-Formular zu einem Zinseszinsberechner umfunktionieren. Der Anwender gibt ein paar Werte ein, JavaScript berechnet etwas und gibt das Ergebnis aus.

All diese Dinge laufen im Web-Browser des Anwenders ab, während eine Web-Seite am Bildschirm angezeigt wird. Es ist keine zusätzliche Kommunikation zwischen Web-Browser und Web-Server erforderlich. Wenn die Web-Seite zum Browser übertragen wurde und sich damit im Arbeitsspeicher des Anwender-Rechners befindet, sind JavaScripts, die in die Web-Seite integriert sind, lokal ausführbar. Damit Programmierer auf dem Rechner des Anwenders keinen Unfug treiben können, ist JavaScript allerdings in seinen Möglichkeiten stark beschränkt. Es ist mit JavaScript beispielsweise nicht möglich, mal eben eine Datei auf dem Rechner des Anwen-

ders auszulesen oder diese gar zu löschen. Die Möglichkeiten von JavaScript sind auf das Umfeld der Web-Seite eingeschränkt, in die so ein Script eingebettet ist.

JavaScripts können Sie ebenso wie CSS Stylesheets direkt innerhalb von HTML-Dateien notieren oder wahlweise als separate Datei einbinden. Damit bezieht sich ein solches JavaScript auf die Seite, in die es eingebunden ist, und kann auf das unmittelbare Umfeld und die Elemente dieser Seite zugreifen. Was den Zugriff auf Elemente einer HTML-Datei betrifft, muss man jedoch einen wichtigen Unterschied machen: Mit klassischem JavaScript, wie es Netscape zunächst einführte, lässt sich nur auf bestimmte Elemente zugreifen, beispielsweise auf Formularelemente. Mittlerweile setzen sich jedoch Ansätze durch, bei denen der Script-Sprachen-Zugriff auf beliebige HTML-Elemente möglich ist, und zwar lesend und schreibend. So wird es beispielsweise möglich, den Text eines Textabsatzes dynamisch auszutauschen, wenn der Anwender an eine bestimmte Stelle klickt. Auf diese Weise werden HTML-Dateien zu richtigen interaktiven Anwendungen – was sich etwa hervorragend für didaktische Zwecke nutzen lässt. Für das dynamische Verändern von am Bildschirm angezeigten Web-Seiten hat sich auch der Begriff **Dynamic HTML** eingebürgert.

Den Zugriff auf beliebige Elemente regelt das **Document Object Model (DOM)**. Dabei handelt es sich nicht um eine Programmiersprache, sondern um ein Schema für Programmiersprachen, das beschreibt, wie auf die Elemente von HTML- und auch auf die Elemente von beliebigen XML-basierten Dokumenten zugegriffen werden kann. Das DOM regelt also beispielsweise, welche Möglichkeiten eine Sprache wie JavaScript bereitstellen muss, um dem Programmierer zu erlauben, den Text einer Überschrift erster Ordnung in einer HTML-Datei zu ermitteln und durch einen anderen Text zu ersetzen. Das DOM ist im Gegensatz zu JavaScript keine konkrete Programmiersprache. Es ist lediglich eine Vorgabe, die in modernen Script-Sprachen für Web-Seiten umgesetzt werden sollte. Denn das DOM wird wie HTML, CSS und JavaScript vom W3-Konsortium standardisiert und hat damit einen normativen Charakter. In den JavaScript-Modulen der verbreiteten Browser wird das DOM allmählich umgesetzt. Der MS Internet Explorer unterstützt DOM-gerechtes JavaScript seit Version 5.x, Netscape seit Version 6.x.

Die Mehrzahl der heutigen Web-Seiten setzt JavaScript in irgendeiner Form ein – die meisten so, wie es gedacht ist, nämlich als Unterstützung für bestimmte Aufgaben, die sich mit HTML einfach nicht lösen lassen. Einige Seiten sind aber auch komplett JavaScript-basiert. Anwender, die in ihrem Browser JavaScript deaktiviert haben (was bei allen Browsern möglich ist), sehen von solchen Seiten rein gar nichts.

## 2.4.2 JavaScript-Informationen

JavaScript ist anders als HTML, CSS oder XML keine unabhängige Technologie, sondern eine von Netscape lizenzierte Programmiersprache. Obwohl JavaScript im MS Internet Explorer genauso funktioniert wie im Netscape-Browser, steckt in Wirklichkeit beim Internet Explorer eine andere Sprache namens **JScript** dahinter. JScript ist die Microsoft-Antwort auf JavaScript, was den Web-Seiten-Entwickler allerdings nicht kümmern muss, denn der JScript-Interpreter des Internet Explorers kommt mit JavaScript zurecht. Als Ersteller von Web-Seiten können Sie also in JavaScript programmieren, was sie wollen, ohne sich um Lizenzfragen kümmern zu müssen. Die Weiterentwicklung von JavaScript liegt jedoch in den Händen des Lizenzinhabers, und nicht in denen einer unabhängigen Organisation wie dem W3-Konsortium. In den Händen des W3-Konsortiums liegt dagegen die Standardisierung des DOM, das eine wichtige Grundlage

für modernes JavaScript ist. Das Dokument für das DOM in der Version 1.0 finden Sie auf den Web-Seiten des W3-Konsortiums.

- Document Object Model (DOM) 1.0 Spezifikation ([www.w3.org/TR/REC-DOM-Level-1/](http://www.w3.org/TR/REC-DOM-Level-1/))

## 2.5 CGI und Perl

### 2.5.1 CGI-Schnittstelle und Perl-Interpreter

Die CGI-Schnittstelle (*Common Gateway Interface – Allgemeine Vermittlungsrechner-Schnittstelle*) ist eine Möglichkeit, Programme oder Scripts im Web bereitzustellen, die von HTML-Dateien aus aufgerufen werden können und selbst HTML-Code erzeugen und an einen Web-Browser senden können. Im Unterschied zu JavaScript werden solche Scripts oder Programme aber nicht nach dem Übertragen einer Web-Seite im Browser des Anwenders ausgeführt, sondern bevor der Browser vom Server Daten übertragen bekommt. Deshalb sind CGI-Scripts oder CGI-Programme auch nur ausführbar, wenn eine HTTP-basierte Kommunikation zwischen Web-Browser und Web-Server stattfindet. CGI ist dabei auf der Server-Seite angesiedelt.

Die Datenverarbeitung geschieht ebenfalls auf dem Server-Rechner. CGI-Programme können auf dem Server-Rechner Daten speichern, zum Beispiel, wie oft auf eine Web-Seite zugegriffen wurde, oder, was ein Anwender in ein Gästebuch geschrieben hat. Ein CGI-Script kann auf dem Server-Rechner auch installierte Datenbanken abfragen. Bei entsprechendem Aufruf kann ein CGI-Programm gespeicherte Daten auslesen und daraus HTML-Code generieren. Dieser »dynamisch« erzeugte HTML-Code wird an den aufrufenden Web-Browser eines Anwenders übertragen und kann dort individuelle Daten in HTML-Form anzeigen, zum Beispiel den aktuellen Zugriffszählerstand einer Web-Seite, die bisherigen Einträge in einem Gästebuch oder Ergebnisse einer Datenbanksuche.

Die CGI-Schnittstelle muss von der Web-Server-Software unterstützt werden. Aus Sicht des Mieters von Speicherplatz auf einem Web-Server steht die CGI-Schnittstelle meistens in Form eines bestimmten Verzeichnisses mit dem Namen *cgi-bin* zur Verfügung. In diesem Verzeichnis können Scripts abgelegt werden, die CGI-Aufgaben übernehmen. Falls Sie unsicher sind, fragen Sie Ihren Provider oder Webmaster, der Ihnen Speicherplatz im Web zur Verfügung stellt, ob er Ihnen eine CGI-Schnittstelle zur Verfügung stellt. Bei preiswerten oder gar kostenlosen Homepage-Vermittlern steht Ihnen normalerweise keine CGI-Schnittstelle zur Verfügung.

Es gibt keine Vorschriften dafür, in welcher Programmiersprache ein CGI-Programm geschrieben ist. Damit das Programm auf dem Server-Rechner ausführbar ist, muss es entweder für die Betriebssystemumgebung des Server-Rechners als ausführbares Programm kompiliert worden sein, oder es muss auf dem Server ein Laufzeit-Interpreter vorhanden sein, der das Programm ausführt. Wenn der Server zum Beispiel ein Linux-Rechner ist, führt er C-Programme aus, die mit einem Linux-C-Compiler zu einer ausführbaren Datei kompiliert wurden. Wenn der Server ein Windows NT-Rechner ist, können CGI-Scripts auch EXE-Dateien sein, die mit Compilern für C, Pascal, Visual Basic usw. erzeugt wurden. Die meisten heutigen CGI-Programme sind jedoch keine kompilierten Programme, sondern lediglich Scripts, die von einem Interpreter beim Aufruf ausgeführt werden. Der bekannteste und beliebteste Interpreter ist dabei der **Perl-Interpreter**.

Perl ist eine Programmiersprache, die eine Mischung aus klassischen Programmiersprachen wie C und Script-Sprachen wie Unix-Shellscrip darstellt. Perl ist extrem leistungsfähig und der Perl-Interpreter hochgradig ausgereift. Perl hat deshalb auf zahllosen Gebieten die Herzen der Programmierer erobert. Unter anderem eben auch auf dem Gebiet der CGI-Programmierung. Aus diesem Grund werden CGI-Programmierung und Perl mittlerweile oft in einem Atemzug genannt und als zusammengehörig dargestellt. Es ist nur wichtig zu wissen, dass die Verbindung von CGI und Perl nichts Naturgegebenes oder Notwendiges ist. CGI ist lediglich eine Norm für eine Programmierschnittstelle, die Web-Server-Software unterstützen sollte, und Perl ist eine universell einsetzbare Script-Sprache, die sich allerdings auf Grund ihrer Charakteristik hervorragend für die CGI-Programmierung eignet.

### 2.5.2 CGI/Perl-Informationen

Für CGI gibt es keinen Standard, der dem Status von HTML, CSS oder XML vergleichbar wäre. Das W3-Konsortium, bei dem es zwar auch eine Arbeitsgruppe gibt, die sich um das HTTP-Protokoll kümmert, klammert die CGI-Schnittstelle aus den HTTP-Dokumenten aus. Die CGI-Schnittstelle wurde mit einem der ersten und seinerzeit erfolgreichsten Web-Server-Produkte eingeführt, nämlich mit dem NCSA-Web-Server (NCSA = *National Center for Supercomputing*, eine akademische Institution). Die Originalspezifikation für die CGI-Schnittstelle von NCSA ist auf den Web-Seiten von NCSA noch verfügbar.

- The CGI Specification ([hoo.hoo.ncsa.uiuc.edu/cgi/interface.html](http://hoo.hoo.ncsa.uiuc.edu/cgi/interface.html))

Allzu viel gehört eigentlich gar nicht dazu – es muss lediglich vom Web-Server unterstützt werden. Der NCSA-Server spielt mittlerweile keine Rolle mehr, die klare Führungsrolle unter den Web-Server-Produkten hat mittlerweile der frei verfügbare Apache Web-Server übernommen. Dieser unterstützt die CGI-Schnittstelle selbstverständlich auch. Perl als Programmiersprache und der Perl-Interpreter sind lizenzfrei und kostenlos verfügbar. Die Einstiegsseite zu allen Belangen rund um Perl ist [www.perl.com](http://www.perl.com).

## 2.6 PHP

### 2.6.1 PHP und HTML

Mit HTML alleine lassen sich keine dynamisch generierten Inhalte erzeugen. Dazu brauchen Sie irgendeine Form von Script-Sprache. JavaScript ist für viele kleinere Fälle recht praktisch, beispielsweise um an einer bestimmten Stelle im Text immer das jeweils aktuelle Tagesdatum auszugeben. Der Vorteil von JavaScript ist, dass es sich problemlos in HTML-Dateien einbetten lässt. Mit CGI/Perl ist wesentlich mehr möglich als mit JavaScript – vor allem all jene Dinge, die eine Datenverarbeitung auf dem Server-Rechner erfordern. Die Datenverarbeitung eines Gästebuchs für ein Web-Projekt beispielsweise muss auf dem Server-Rechner stattfinden. Der Grund ist, dass Daten zentral gespeichert werden müssen, damit sie allen Seitenbesuchern zur Verfügung stehen – im Fall des Gästebuchs die vorhandenen Gästebucheinträge. Der Nachteil von CGI und Perl ist, dass Perl-Scripts separate Dateien sind, die aufgrund der Charakteristik der CGI-Schnittstelle meistens in ganz anderen Verzeichnissen abgelegt werden müssen als die HTML-Dateien des Web-Projekts. Ein weiteres Problem ist, dass Perl als Universalsprache aus-

gelegt ist und nicht speziell für dynamische Web-Seiten entwickelt wurde. Manche Dinge, die sich ein Entwickler von dynamischen Web-Seiten wünscht, sind in Perl nur schwer und über Umwege zu erreichen. Gute Perl-Programmierer bekommen zwar jedes Problem in den Griff – aber der Weg zum guten Perl-Programmierer ist weit und lang.

An diesen »Problemzonen« von Perl setzt **PHP** an. Die Abkürzung steht für *Hypertext Preprocessor*. Das Konzept dahinter ist, dass PHP-Code ähnlich wie JavaScript direkt in HTML-Dateien an einer dafür geeigneten Stelle notiert werden kann. Wenn die HTML-Datei dann im Web abgelegt ist und von einem Web-Browser aufgerufen wird, erkennt der Web-Server, der die Datei zum Browser übermittelt, aufgrund bestimmter Konventionen, dass es sich nicht um eine gewöhnliche HTML-Datei handelt, sondern um eine HTML-Datei mit eingebettetem PHP-Code. Eine solche Datei lässt er dann zunächst von dem serverseitig installierten PHP-Interpreter verarbeiten. Dieser liest in der HTML-Datei die PHP-Code-Passagen aus, führt den Code aus und erzeugt daraus den endgültigen HTML-Code, der schließlich an den Browser gesendet wird.

PHP kann alles, was mit CGI und Perl auch möglich ist. Und manches ist mit PHP durchaus einfacher als mit CGI und Perl, weil der PHP-Interpreter viel stärker als etwa der Perl-Interpreter auf aktuelle Belange des Web-Publishing ausgerichtet ist. So ist es beispielsweise mit PHP auch möglich, PDF-Dateien dynamisch zu generieren, um druckreife Daten an den Browser zu senden. Mit Perl geht so etwas zwar auch, doch dazu muss der Programmierer sich erst mal nach einem geeigneten Modul umsehen, dieses installieren und sich dann mit der moduleigenen Dokumentation beschäftigen. Beim PHP-Interpreter ist dagegen alles inklusive und fest eingebaut. Allerdings platzt der PHP-Interpreter aufgrund seines Feature-Umfangs allmählich aus allen Nähten und wird durch die »Alles-inklusive«-Philosophie nicht gerade schneller – auch wenn er mittlerweile die gleiche interne Technik wie der Perl-Interpreter verwendet, nämlich Scripts erst einmal zu kompilieren und sie dann auszuführen. Letztendlich aber bestimmen immer eine ganze Reihe von Faktoren mit darüber, wie performant eine dynamische Web-Seite wirklich ist – unter anderem die Anzahl der Seitenaufrufe, die Leistungsfähigkeit der Server-Hardware, die Netzanbindung des Servers usw. In der Praxis hat sich PHP jedenfalls bestens bewährt und wird von vielen großen Web-Angeboten mit Erfolg eingesetzt.

## 2.6.2 PHP-Informationen

PHP und der PHP-Interpreter sind frei verfügbar und lizenzfrei benutzbar. Entwickelt werden Sprache und Interpreter von der so genannten PHP Group, einem Projektzusammenschluss von Programmierern. Die Einstiegsseite zu allen Belangen rund um PHP ist [www.php.net/](http://www.php.net/).

## 2.7 ASP

### 2.7.1 ASP und HTML

ASP steht für **Active Server Pages** (*Aktive Server-Seiten*) und stellt ähnlich wie PHP eine Alternative zu CGI/Perl dar. Auch bei ASP geht es also darum, serverseitig dynamische Web-Seiten zu erzeugen, beispielsweise, um Suchtreffer, die sich aufgrund einer Datenbankabfrage ergeben, in HTML-Form aufgelistet an den aufrufenden Browser zu senden. Ähnlich wie PHP-Code können Sie Script-Code bei ASP direkt in HTML einbetten – es sind also nicht wie bei Perl und CGI

separate Scripts erforderlich, die in speziellen Verzeichnissen abgelegt werden müssen. Im Unterschied zu PHP ist ASP jedoch stärker an die Welt von Microsoft und Windows gekoppelt. Zwar gibt es ASP durchaus auch für Linux und andere Unix-basierte Systeme, und neben Microsoft Web-Servern werden auch der Apache Server und viele andere unterstützt. Doch die ASP-Integration ist für Web-Server von Microsoft optimiert und wird daher vorwiegend unter Windows NT eingesetzt.

ASP ist im Gegensatz zu PHP nur eine »Umgebung«, keine eigene Script-Sprache. Als Script-Sprachen kommen vor allem die Microsoft-eigenen Sprachen JScript und VBScript (Visual Basic Script) in Frage. Aber auch der Einsatz von Perl für ASP ist möglich – und hier zeigt sich denn auch wieder die universelle Verwendbarkeit von Perl, das etwa im Gegensatz zu PHP nicht an bestimmte Umgebungen gebunden ist. VBScript ist allerdings die Standard-Script-Sprache für ASP.

Das ASP-Prinzip ist das gleiche wie bei PHP. Aufgrund bestimmter Konventionen erkennt der Web-Server, dass eine HTML-Datei eine ASP-Datei ist. Der Server mit integrierter ASP-Umgebung führt dann den in der Datei enthaltenen Script-Code aus und sendet am Ende den dadurch erzeugten, gesamten HTML-Code an den aufrufenden Browser.

### 2.7.2 ASP-Informationen

Die ASP-Umgebung ist lizenz- und kostenpflichtig. Vertrieben wird ASP von Sun Chili!Soft ([www.chilisoft.com/](http://www.chilisoft.com/)).

## 2.8 Java

### 2.8.1 Java und HTML

Java ist eine von Sun Microsystems entwickelte, vollkommen plattformunabhängige Programmiersprache. Die Sprache lehnt sich in Aufbau und Syntax an C/C++ an. Java wurde zunächst hauptsächlich als Programmiersprache für Internet-Anwendungen gefeiert. Nach einigen Jahren zeichnet sich jedoch ab, dass Java zwar für alle möglichen Zwecke gut geeignet ist, dass es aber im Web nicht die Dominanz erreicht hat, die von Sun ursprünglich mal erhofft wurde. Im Web wird Java heute vor allem im Zusammenhang mit Online-Banking, Online-Broking, web-basierten Chats und für didaktische Zwecke eingesetzt. Im Bereich der Animationen, Effekte, Online-Spiele usw. wird Java dagegen immer stärker von Flash verdrängt.

Java zeichnet sich u.a. durch folgende Eigenschaften aus:

- **Objekt-Orientierung:** Java ist streng objektorientiert und stellt dem Programmierer eine Fülle elementarer und komplexer Objekte bereit.
- **Sichere Speicherverwaltung:** Unter Java ist die Arbeitsspeicherverwaltung aus Programmiersicht wesentlich unkritischer als beispielsweise unter C. Es gibt keine durch Zeiger adressierten Speicherbereiche von unbestimmter Größe, sondern nur Speichervektoren von fest definierter Länge. Die fehlerarme Speicherverwaltung ist besonders wichtig, da Java-Programme ohne Portierungsaufwand in verschiedenen Rechnerumgebungen ablaufen sollen.

- **Plattformunabhängige Programmdateien:** Java-Programme werden wie herkömmliche Programme zu Objektcode kompiliert, jedoch nicht in eine bestimmte Prozessor- oder Betriebssystemumgebung eingebunden, also nicht »gelinkt«. Java-Programme laufen daher auf allen Plattformen, wenn ein Java-Objektcode-Interpreter installiert ist. Web-Browser, die Java ausführen, starten ihre eigene »Java-Konsole« zu diesem Zweck.

Java-Programme, die für den Einsatz im Internet gedacht sind, heißen **Applets**. Sie können Java-Applets in HTML-Dateien so referenzieren, dass die Anwendung innerhalb der Web-Seite in einem dazu geeigneten Bereich erscheint. Die Bildschirmausgaben bzw. die Interaktionen zwischen Anwender und Programm finden also im Anzeigefenster des WWW-Browsers statt. Applets sind eine Sonderform von Java-Programmen, die in ihren Möglichkeiten beschränkt sind. Auf diese Weise wird sichergestellt, dass Java-Applets nicht ohne das Einverständnis des Anwenders irgendetwas auf dessen Rechner tun können, z.B. Dateien verändern oder löschen. Java-Applets laufen also in einer so genannten **Sandbox**, einem Sicherheitskäfig.

Innerhalb des Client-Server-Modells im Web kann Java auf beiden Seiten vorkommen. Applets, die im Browser ausgeführt werden, sind clientseitig, laufen also auf dem Rechner des Anwenders. Häufig kommunizieren solche Applets aber über eigene Protokolle online mit entsprechenden, in Java programmierten »Gegenstellen« auf einem Server. Bei einer Anwendung wie einem Chat beispielsweise gibt es einerseits ein Applet, das im Browser läuft und dem Anwender die Chat-Oberfläche präsentiert – doch damit tatsächlich ein Mehrpersonen-Chat möglich ist, muss natürlich auch auf dem Server irgendetwas laufen, was die Chat-Teilnehmer verwaltet und in den Client-Anwendungen anzeigt. Hinter Java-Applets verbergen sich deshalb häufig verteilte Anwendungen, die aus einem Applet und zugehörigen Server-Programmen bestehen.

Um selbst ausführbare Java-Applets erstellen zu können, brauchen Sie einen Texteditor, um den Java-Quellcode zu generieren, und das Java Entwickler-Kit von Sun. Das Entwickler-Kit enthält alle Klassenbibliotheken der Programmiersprache Java, einen Applet-Viewer und einen Compiler. Neben einem Compiler, der speziell Java-Applets zum Einsatz im WWW erzeugt, gibt es im Java Developer's Kit aber noch einen Compiler, der Internet-unabhängige Programme für beliebige PCs oder andere Softwareträger erzeugt. Dieser Compiler erlaubt alle Möglichkeiten, um vom Webunabhängige Software zu erstellen. Für größere Softwareprojekte mit Java gibt es ausgereifte Java-Entwicklungsumgebungen, ähnlich wie für C++, Visual Basic oder Delphi.

## 2.8.2 Java-Informationen

Das Java-Entwickler-Kit ist frei für den persönlichen Gebrauch und für nichtkommerzielle Zwecke. Wenn Sie kommerzielle Anwendungen mit Java erstellen wollen, müssen Sie für die Compiler-Software eine kostenpflichtige Lizenz erwerben. Über die folgende Seite finden Sie im Web den Einstieg zu Software und Dokumentation rund um Java:

- Java-Entwickler-Seiten (Sun, [java.sun.com/](http://java.sun.com/))

## 2.9 ActiveX

### 2.9.1 ActiveX und HTML

ActiveX ist eine von Microsoft eingeführte Technologie für ausführbaren Programmcode auf Web-Seiten mit dem Anspruch einer Alternative oder Konkurrenz zu Java. ActiveX ist dabei ein Überbegriff für verschiedene Softwarekomponenten. Alle diese Komponenten basieren auf dem so genannten **Component Object Model (COM)**. ActiveX ist jedoch kein Internet-Standard im herkömmlichen Sinn, sondern der Versuch, spezifische Eigenschaften des MS-Windows-Betriebssystems für Web-Seiten nutzbar zu machen. So ist es mit Hilfe von ActiveX beispielsweise möglich, Eingaben aus HTML-Formularen einer Web-Seite über die OLE-Schnittstelle von Windows direkt in eine Excel-Tabelle oder eine Access-Datenbank einzulesen (oder umgekehrt). ActiveX-Controls benutzen nämlich die OLE-Schnittstelle von Windows, um mit ihrer Umgebung zu kommunizieren. Microsoft bemüht sich zwar um Lösungen, die sich auch auf Macintosh- und Unix-Rechner übertragen lassen. Doch letztendlich ist ActiveX für MS-Windows optimiert, und das mit Absicht.

Ein wesentlicher Bestandteil von ActiveX sind die so genannten ActiveX-Controls. Solche Controls sind Programme oder Programm-Module, die sich in HTML-Dateien als Objekt einbinden lassen, ähnlich wie Java-Applets. Der Programmcode wird im Arbeitsspeicher des Client-Rechners (also des Anwenders, der eine Web-Seite aufruft) ausgeführt. Normalerweise wird dabei auch eine so genannte ActiveX-Layout-Datei auf dem Rechner des Anwenders installiert. Mit ActiveX-Controls lassen sich alle Arten von Anwendungen usw. realisieren.

Es gibt keine festen Vorschriften dafür, in welcher Programmiersprache ActiveX-Code geschrieben ist. Der Compiler muss jedoch das Component Object Model (COM) unterstützen. Um ActiveX-Controls zu erstellen, brauchen Sie also keine bestimmte Software, aber Sie brauchen einen COM-fähigen Compiler. Das kann Visual Basic sein, Microsoft C++ usw.

ActiveX wird derzeit nur vom MS Internet Explorer direkt ausgeführt. Für Netscape gibt es ein ActiveX-Plug-In zum Downloaden. Bei Netscape-Anwendern, die das Plug-In installiert haben, sind ActiveX-Controls dann auch ausführbar.

Diskutiert wird häufig über das Sicherheitskonzept von ActiveX. Wenn Sie als Anwender erlauben, dass ein ActiveX-Control auf Ihren Rechner geladen wird, kann dieses Programm auf dem Rechner so ziemlich tun und lassen was es will. Es gibt also keine »Sandbox«-Beschränkungen bei ActiveX, sondern nur eine »Vertrauensbarriere« (nämlich die Abfrage des Web-Browsers, ob der Anwender mit der Ausführung des Programms auf seinem Rechner einverstanden ist oder nicht). Dadurch ist mit ActiveX aus Programmiersicht zwar viel mehr realisierbar als mit Java-Applets, doch aus Anwendersicht sind ActiveX-Module tendenziell unsicherer als Java-Applets.

### 2.9.2 ActiveX-Informationen

Auf den Web-Seiten von Microsoft ([www.microsoft.com/](http://www.microsoft.com/)) können Sie nach Informationen zu ActiveX suchen. Microsoft bietet sie an verschiedenen Stellen an. Über die Domain [activex.com](http://activex.com) ([www.activex.com/](http://www.activex.com/)) werden Sie auf eine Adresse weitergeleitet, an der Sie geeignete Software und vorhandene ActiveX-Controls finden. In diesem Buch wird die ActiveX-Technologie nicht weiter behandelt.



## 2.10 Flash

### 2.10.1 Flash und HTML

Flash bietet die Möglichkeit, Multimedia-Effekte auf Web-Seiten zu bringen, aber auch Anwendungen wie Spiele, Simulationen oder Navigationsunterstützung für Web-Seiten.

Flash ist jedoch der jüngste Hype und der neueste Vertreter der Gattung »ich mach HTML tot« – zumindest ist es von jenen so hochstilisiert worden, die sich mit HTML und den offenen Standards noch nie abfinden konnten. Da hinter Flash ein kommerzielles Softwareprodukt steckt, das in keiner Weise mit den offenen Internet-Standards vergleichbar ist, werden also Äpfel mit Birnen verglichen. Genauso könnte man sagen »ich brauche kein HTML für Tabellen, ich habe schließlich MS Excel«. Und es spricht nichts dagegen, eine Excel-Datei als Objekt in HTML einzubinden oder einfach so ins Web zu stellen und die Besucher auf diese Datei zu lenken – bei installiertem Excel oder Excel-Viewer wird der Browser dann brav die Anwendung öffnen und die Datei darin anzeigen. Nicht anders ist es bei Flash. Es handelt sich um ein Binärformat, das Sie wahlweise als Multimedia-Objekt in HTML einbinden können, oder Sie bieten Flash-Dateien direkt an – die neueren Browser haben ein Plug-In für Flash und können Flash direkt anzeigen. Mit Internet-Standards hat das alles jedoch nichts zu tun, und wenn Macromedia, dem Softwarehersteller von Flash, morgen einfällt, dass Flash eingestampft oder völlig umgekrempelt wird, dann ist das eben so, und niemand kann etwas dagegen tun.

Flash steht vor allem dort hoch im Kurs, wo man der Meinung ist, auf einer Web-Seite müsse es zugehen wie im Fernsehen. Dauernd irgendetwas in Bewegung, schicke Überblenden, Licht-Effekte, Geräusche, Musik usw. Manche Kreativlinge unterliegen diesem Glauben, aber noch mehr die Marketing-Abteilungen von Firmen, die bei Agenturen ihren Web-Auftritt gestalten lassen und dafür am Bildschirm etwas sehen wollen, von dem sie glauben, es sei besonders schwierig und atemberaubend. Dabei schaden die – von der Mehrheit der Seitenbesucher übrigens als sehr nervig empfundenen – Effekthaschereien, für die Flash vorwiegend eingesetzt wird, dieser Technologie mehr als sie ihr nutzen. Denn eigentlich ist Flash durchaus sinnvoll einsetzbar. Flash basiert auf bewegter Vektorgrafik. Beim Erstellen der so genannten **Flash-Movies** arbeiten Sie mit der Flash-Autoren-Software, mit der Sie grafische Darstellungen, Text und eingebundene Elemente wie Pixelgrafiken oder Sound an einer Zeitleiste positionieren. Dadurch bestimmen Sie einen zeitlichen Ablauf. Flash kann Übergänge zwischen zwei Zuständen auf der Zeitleiste berechnen. Angenommen, Sie haben bei 0 Sekunden einen blauen Punkt und bei 1 Sekunde an der gleichen Stelle einen roten Punkt, dann können Sie für den Punkt einen fließenden Farbübergang von Blau nach Rot definieren, der 1 Sekunde lang dauert. Nicht jedes »Movie« hat jedoch Anfang und Ende. Ebenso sind Schleifen-Effekte möglich, durch die sich mit Flash dialogorientierte Anwendungen wie Spiele oder Navigationshilfen realisieren lassen. Flash bietet auch webspezifische Schnittstellen an. So ist es problemlos möglich, in Flash anklickbare Verweise zu anderen Web-Seiten einzubrauen. Im didaktischen Bereich ist Flash beispielsweise gut einsetzbar, weil es sich hervorragend zur Visualisierung von Abläufen und Zusammenhängen eignet. Vereinzelt wird es auch für solche sinnvollen Zwecke eingesetzt – doch im Web dominieren leider vorwiegend die dümmlich daherfliegenden Schriftzüge und Blitz-Effekte.

Für anspruchsvollere Anwendungen reicht die visuelle Oberfläche der Flash-Autoren-Software alleine oft nicht aus. Flash hat deshalb zusätzlich eine integrierte Programmiersprache, die es Entwicklern erlaubt, komplexere Aufgaben zu lösen.

Alles in allem ist Flash also ein mächtiges Werkzeug, um Inhalte zu visualisieren. Nicht unerwähnt bleiben sollte jedoch, dass es auch frei verfügbare, offen dokumentierte und letztlich noch leistungsfähigere Internet-Standards für das gibt, was Flash leistet. So werden beim W3-Konsortium zwei XML-basierte Sprachen namens SVG und SMIL standardisiert, die in Verbindung mit JavaScript und DOM ähnliche Möglichkeiten eröffnen. Doch leider werden die modernen Browser mit Flash-Plug-Ins ausgeliefert, sind aber noch nicht ohne weiteres in der Lage, SVG oder SMIL zu verarbeiten. Und solange es kein Autorenwerkzeug dafür gibt, das auch solche Web-Entwickler anspricht, die keinen Quelltext am Bildschirm ertragen können, werden es diese Standards gegen Erscheinungen wie Flash schwer haben.

### 2.10.2 Flash-Informationen

Auf den Web-Seiten von Macromedia ([www.macromedia.com/](http://www.macromedia.com/)) können Sie nach Informationen zu Flash suchen. In allen großen Suchmaschinen im Web finden Sie zahlreiche Web-Angebote, die sich mit Flash auseinandersetzen. Die Autoren-Software, die zum Erstellen von Flash-Movies erforderlich ist, ist kosten- und lizenzpflichtig. Die Flash-Player-Software, die auch als Plug-In bei vielen Browsern mit ausgeliefert wird, ist dagegen frei verwendbar. In diesem Buch wird Flash nicht weiter behandelt.

## 3 Allgemeine Regeln für HTML

### 3.1 Textauszeichnung

#### 3.1.1 Elemente und Tags in HTML

HTML-Dateien bestehen aus Text. Zur Textauszeichnung gibt es bestimmte Zeichen aus dem normalen Zeichenvorrat. Der Inhalt von HTML-Dateien steht in **HTML-Elementen**. HTML-Elemente werden durch so genannte **Tags** markiert. Fast alle HTML-Elemente werden durch ein einleitendes und ein abschließendes Tag markiert. Der Inhalt dazwischen ist der »Gültigkeitsbereich« des entsprechenden Elements. Tags werden in spitzen Klammern notiert. Ein Beispiel:

```
<h1>HTML - die Sprache des Web</h1>
```

Das Beispiel zeigt eine Überschrift 1. Ordnung. Das einleitende Tag `<h1>` signalisiert, dass eine Überschrift 1. Ordnung folgt (*h = heading = Überschrift*). Das abschließende Tag `</h1>` signalisiert das Ende der Überschrift. Ein abschließendes Tag beginnt mit einem Schrägstrich (/).

**Beachten Sie:** Bei herkömmlichem HTML spielt es keine Rolle, ob die Elementnamen in den Tags in Klein- oder Großbuchstaben notiert werden. Dort bedeuten z.B. `<h1>` und `<H1>` das Gleiche. In der neueren HTML-Variante, in XHTML, müssen die Elementnamen dagegen klein geschrieben werden. Das W3-Konsortium empfiehlt zwar für HTML aus Gründen der Lesbarkeit, Namen von Elementen groß zu schreiben. Im Hinblick auf eine spätere Deklaration eines Dokuments als XHTML-Dokument ist diese Idee jedoch nicht unbedingt so gut und es ist besser, von vorneherein alle Elementnamen in Kleinbuchstaben zu schreiben.

Es gibt auch einige Elemente mit Standalone-Tags, d.h. Elemente, die keinen Inhalt haben und deshalb nur aus einem Tag bestehen statt aus Anfangs- und End-Tag. Ein Beispiel:

```
Eine Zeile, ein manueller Zeilenumbruch<br>
und die nächste Zeile
```

Am Ende der ersten Zeile signalisiert `<br>`, dass ein manueller Zeilenumbruch eingefügt werden soll (*br = break = Umbruch*).

**Beachten Sie:** Wenn Sie XHTML-gerecht schreiben wollen, müssen Sie Elemente mit Standalone-Tags anders notieren: Anstelle von `<br>` müssen Sie dann `<br />` notieren – also den Elementnamen mit einem abschließenden Schrägstrich. Alternativ dazu können Sie auch `<br></br>` notieren, also ein Element mit Anfangs- und End-Tag, aber ohne Inhalt. Mehr darüber erfahren Sie im Kapitel 14 (*XHTML und HTML*).

### 3.1.2 Verschachtelung von Elementen

Elemente können ineinander verschachtelt werden. Auf diese Weise entsteht eine hierarchische Struktur. Komplexere HTML-Dateien enthalten sehr viele Verschachtelungen. Deshalb sprechen Fachleute auch von **strukturiertem Markup**. Ein Beispiel:

```
<h1><i>HTML</i> - die Sprache des Web</h1>
```

Das `i`-Element steht für schräg gestellte Schrift (*italic* = *kursiver Text*). Der Text zwischen `<i>` und `</i>` wird als kursiv interpretiert, abhängig von der eingestellten Schriftart und Schriftgröße für die Überschrift 1. Ordnung.

### 3.1.3 Attribute in Tags

Einleitende Tags und Standalone-Tags können zusätzliche Angaben enthalten. `align="center"` bewirkt, dass der Text zentriert ausgerichtet wird (*align* = *Ausrichtung*, *center* = *zentriert*). Ein Beispiel:

```
<h1 align="center">HTML - die Sprache des Web</h1>
```

Es gibt folgende Arten von Attributen in HTML-Elementen:

- Attribute mit Wertzuweisung, wobei es bestimmte erlaubte Werte gibt, z.B. bei `<h1 align="center">` (Überschrift 1. Ordnung zentriert ausgerichtet – hier sind nur die Werte `left`, `center`, `right` und `justify` erlaubt).
- Attribute mit freier Wertzuweisung, wobei jedoch ein bestimmter Datentyp oder eine bestimmte Konvention erwartet wird, z.B. `<style type="text/css">` (Bereich für Stylesheets definieren – hier wird ein so genannter Mime-Typ als Wert erwartet, und Mime-Typen haben immer den Aufbau *Typ/Untertyp*). Oder `<table border="1">` (Tabelle mit Rahmen von 1 Pixel Stärke – hier wird eine numerische Angabe erwartet.)
- Attribute mit freier Wertzuweisung ohne weitere Konventionen, z.B. `<p title="Aussage mit Vorbehalt">` – hier kann ein ganzer Text zugewiesen werden.
- Allein stehende Attribute, z.B. `<hr noshade>` (Trennlinie ohne Schatten). Allein stehende Attribute gibt es allerdings nur in herkömmlichem HTML. Wenn Sie XHTML-gerecht schreiben wollen, müssen Sie `<hr noshade="noshade">` notieren. Mehr darüber erfahren Sie im Kapitel 14.1.9 (*XHTML und HTML*).

Alle Werte, die Sie Attributen zuweisen, müssen in Anführungszeichen stehen. Die meisten Browser nehmen es zwar nicht übel, wenn die Anführungszeichen fehlen, doch seit dem HTML-Standard 4.0 sind die Anführungszeichen klipp und klar vorgeschrieben, und wer ordentliches HTML schreiben will, sollte sich daran halten.

Wie bei Elementnamen, so gilt auch bei Attributnamen: Bei herkömmlichem HTML spielt es keine Rolle, ob die Attributnamen in Klein- oder Großbuchstaben notiert werden. In der neueren HTML-Variante, in XHTML, müssen die Attributnamen dagegen klein geschrieben werden. Bei den Wertzuweisungen an Attribute kann Groß- und Kleinschreibung abhängig von der Art des Wertes unterschieden werden oder auch nicht.

Neben Attributen, die nur in bestimmten HTML-Elementen vorkommen können, gibt es auch so genannte **Universalattribute**, die in vielen bzw. fast allen HTML-Elementen erlaubt sind. Ein Beispiel:

```
<p id="Einleitung">Text</p>
```

Das Beispiel definiert einen Textabsatz mit den HTML-Tags `<p>` und `</p>`. Im einleitenden `<p>`-Tag wird ein Universalattribut notiert, nämlich das Attribut `id=`. Damit können Sie dokumentweit eindeutige Namen für einzelne HTML-Elemente vergeben. Einzelheiten über solche Attribute erfahren Sie im Abschnitt 13.1.

### 3.1.4 HTML-Parser

Unter einem HTML-Parser versteht man eine Software, die HTML-Auszeichnungen erkennt und in strukturierten Text umsetzt. Jeder Web-Browser verfügt über einen HTML-Parser, um überhaupt mit HTML klarzukommen. Solche HTML-Parser werden nun leider auf den meisten Web-Seiten mit Syntaxfehlern in der Textauszeichnung konfrontiert. Oft sind es kleinere, nicht allzu tragische Fehler, doch es gibt auch viele Web-Seiten, deren HTML-Quelltext nur das Prädikat »ungenügend« verdient, weil darin übelste Verunstaltungen der HTML-Regeln vorkommen. Strenge Parser, die genau gegen die HTML-Regeln prüfen, müssten die Umsetzung solcher Web-Seiten eigentlich abbrechen, und anstelle der Seite würden die Browser dann nur eine lapidare Fehlermeldung anzeigen. Da ein solcher Browser am breiten Markt jedoch keine Chance hätte, weil er kaum eine bekannte Web-Seite anzeigen würde, sind die HTML-Parser der heute verbreiteten Browser ziemlich gutmütige Wesen, die so ziemlich alles fressen, was ihnen vorgesetzt wird, und irgendetwas daraus machen, meistens sogar durchaus das, was der Autor der Web-Seite erreichen wollte. Am weitesten in dieser Kunst hat es der HTML-Parser des MS Internet Explorer gebracht. Das hat dem Internet Explorer einerseits den Ruf beschert, »am besten« HTML zu können, aber Fachleute rümpfen aus dem gleichen Grund gerne die Nase über diesen Browser mit dem Argument, dass er durch sein Verhalten das schlampige und fehlerhafte Kodieren von HTML nur fördere.

Angesichts der wachsenden Komplexität der verschiedenen Sprachen, also HTML in Verbindung mit eingebettetem CSS, JavaScript, PHP usw., wird es immer wichtiger, die Syntaxregeln von HTML einzuhalten. Diese Regeln finden Sie in der HTML-Referenz im Anhang genauer beschrieben.

## 3.2 Grundgerüst einer (X)HTML-Datei

Eine gewöhnliche HTML-Datei besteht grundsätzlich aus folgenden Teilen:

- Dokumenttyp-Angabe (Angabe zur verwendeten HTML-Version)
- Header (Kopfdaten, z.B. Angaben zu Titel u.ä.)
- Body (Körper – anzuzeigender Inhalt, also Text mit Überschriften, Verweisen, Grafikreferenzen usw.)

### 3.2.1 Grundgerüst einer HTML-Datei

Das folgende Listing zeigt das Schema einer vollständigen HTML-Datei mit allen notwendigen Tags und Angaben, aber ohne Inhalt:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

</body>
</html>
```

Die erste Zeile sieht für Anfänger zunächst verwirrend aus. Diese etwas komplizierte Angabe ist eine Dokumenttyp-Angabe. Einzelheiten dazu weiter unten bei Abschnitt 3.2.3 (*Dokumenttyp-Angaben*).

Der gesamte übrige Inhalt einer HTML-Datei wird in die Tags `<html>` bzw. `</html>` eingeschlossen. Das `html`-Element wird auch als **Wurzelement** einer HTML-Datei bezeichnet. Hinter dem einleitenden HTML-Tag folgt das einleitende Tag für den Kopf `<head>`. Zwischen diesem Tag und seinem Gegenstück `</head>` werden die Kopfdaten einer HTML-Datei notiert. Die wichtigste dieser Angaben ist der Titel der HTML-Datei, markiert durch `<title>` bzw. `</title>`. Unterhalb davon folgt der Textkörper, markiert durch `<body>` bzw. `</body>`. Dazwischen wird dann der eigentliche Inhalt der Datei notiert, also das, was im Anzeigefenster des WWW-Browsers angezeigt werden soll.

**Beachten Sie:** Wenn Sie Frames (mehrere Bildschirmfenster) einsetzen wollen, sieht das Grundgerüst von Dateien, in denen ein Frameset definiert wird, anders aus. Dies wird im Abschnitt 9.1.2 *Grundgerüst einer HTML-Datei mit Framesets* beschrieben. Befassen Sie sich mit Frames aber erst, wenn Sie mit den Grundlagen von HTML schon etwas vertrauter sind.

### 3.2.2 Grundgerüst einer XHTML-Datei

Wenn Sie XHTML-gerechtes HTML schreiben wollen, sieht das Grundgerüst ähnlich aus. Nur am Anfang ist einiges anders. Das Schema:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Text des Titels</title>
</head>
<body>
```

```
</body>
</html>
```

Noch vor der Dokumenttyp-Angabe sollte bei XHTML-Dateien der Bezug zu XML hergestellt werden. Dazu dient die erste Zeile mit den Fragezeichen hinter der öffnenden spitzen Klammer und der schließenden spitzen Klammer. Notieren Sie diese Zeile so wie im Beispiel angegeben. Es handelt sich um eine so genannte **XML-Deklaration**.

Bei der Dokumenttyp-Angabe muss ein für XHTML gültiger Dokumenttyp angegeben werden. Einzelheiten dazu im folgenden Abschnitt 3.2.3 über Dokumenttyp-Angaben. Im einleitenden `<html>`-Tag muss der verwendete XML-Namensraum mit einem Attribut `xmlns=` angegeben werden. Benutzen Sie die Angabe wie im obigen Beispiel.

Damit ist die Datei als XHTML-Datei deklariert. Der weitere Quelltext ist normales HTML. Allerdings müssen Sie in diesem Fall die Unterschiede zwischen XHTML und HTML beachten (Kapitel 14). Befassen Sie sich mit diesen Unterschieden aber erst, wenn Sie schon etwas vertrauter mit HTML sind.

### 3.2.3 Dokumenttyp-Angaben

HTML ist innerhalb der Familie der Auszeichnungssprachen nur eine von vielen, wenn auch die prominenteste. HTML selbst hat außerdem bereits eine mehrjährige Geschichte und ist in verschiedenen, recht unterschiedlichen Versionen normiert worden. Mit der Dokumenttyp-Angabe bestimmen Sie, welche Auszeichnungssprache in welcher Version Sie verwenden. Eine auslesende Software, etwa ein Web-Browser, kann sich an dieser Angabe orientieren.

Die Regeln für HTML sind mit Hilfe von SGML formuliert, die Regeln für XHTML mit Hilfe von XML. Nach den Regeln einer SGML- oder XML-basierten Auszeichnungssprache ist eine HTML-Datei oder eine XHTML-Datei erst dann eine **gültige (valide)** Datei, wenn sie einen bestimmten Dokumenttyp angibt und sich dann innerhalb des restlichen Quelltextes genau an die Regeln hält, die in diesem Dokumenttyp definiert sind. Denn hinter jeder Dokumenttyp-Angabe stecken so genannte **Dokumenttyp-Definitionen (DTD)**. Auch für HTML gibt es solche Dokumenttyp-Definitionen. Dort ist geregelt, welche Elemente ein Dokument vom Typ HTML enthalten darf, welche Elemente innerhalb von welchen anderen vorkommen dürfen, welche Attribute zu einem Element gehören, ob die Angabe dieser Attribute Pflicht ist oder freiwillig usw.

Als HTML-Anfänger mag Ihnen der Aufwand, der da betrieben wird, vielleicht nicht ganz einleuchten. Doch genau diese Dokumenttypen, mit deren Hilfe sich Regeln für Sprachen wie HTML genau definieren lassen, sind ein riesiger Fortschritt in der EDV. Denn nur so lässt sich das Konzept der softwareunabhängigen, aber regelgerechten Dateiformate konsequent durchsetzen. Ohne den Bezug auf die offiziellen Regeln wären Sprachen wie HTML unverbindliche Konventionen, die schnell in Dialekten verwässern würden. Das ist genauso wie bei natürlichen Sprachen: Ohne eine gewisse Regelung der Rechtschreibung würde sich eine Schriftsprache im Laufe der Zeit in so verschiedene Richtungen entwickeln, dass am Ende kaum jemand mehr verstehen kann, was der andere meint. Da Software außerdem noch viel dümmer ist als Menschen und viel genauere Vorgaben benötigt, um zu »verstehen«, was man ihr mitteilt, ist das Einhalten von Regeln dort sogar noch wesentlich wichtiger.

Beispiel einer Dokumenttyp-Angabe:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

Notieren Sie die Dokumenttyp-Angabe am Anfang der HTML-Datei vor dem einleitenden `<html>`-Tag. Hinter der startenden spitzen Klammer folgt ein Ausrufezeichen. Dahinter folgt die Angabe `DOCTYPE HTML PUBLIC`. Das bedeutet, dass Sie sich auf die öffentlich verfügbare HTML-DTD beziehen. Die folgende Angabe, die in Anführungszeichen steht, ist wie folgt zu verstehen:

W3C ist der Herausgeber der DTD, also das W3-Konsortium. Eine Angabe wie `DTD HTML 4.01 Transitional` bedeutet, dass Sie in der Datei den Dokumenttyp »HTML« verwenden, und zwar in der Sprachversion 4.01 und deren Variante »Transitional«. Das `EN` ist ein Sprachenkürzel und steht für die Sprache, in diesem Fall Englisch. Die Angabe bezieht sich darauf, in welcher natürlichen Sprache die Element- und Attributnamen der Tag-Sprache definiert wurden, nicht auf den Inhalt Ihrer Datei. Benutzen Sie also immer `EN`, da die Namen von HTML-Elementen und -Attributen auf der englischen Sprache basieren. Die Schrägstriche notieren Sie so wie im obigen Beispiel.

Ferner enthält die Dokumenttyp-Angabe eine Web-Adresse. Die Angabe dieser Adresse ist nicht zwingend nötig. Eine Angabe wie:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN">
```

ist also ebenfalls erlaubt. Über die angegebene Web-Adresse kann eine auslesende Software die Dokumenttyp-Definitionen (DTD) aufrufen und in den darin notierten Regeln »nachgucken«. Die meisten heutigen Browser tun das bei HTML allerdings nicht, weil sie ohnehin bei vielen Web-Seiten mit übelsten Sprachverunstaltungen rechnen müssen und die Fähigkeit, auch nicht regelgerechtes HTML ordentlich auf den Bildschirm zu bringen, zugunsten der »reinen Lehre« opfern. Das eigentlich gewünschte Verhalten ist allerdings, dass ein Browser dann, wenn eine HTML-Datei eine Dokumenttyp-Angabe enthält, auch auf der Einhaltung der Regeln dieses Dokumenttyps besteht. Bei XML-Dokumenten ist es durchaus üblich, dass ein Parser bei einem Regelverstoß abbricht – anstelle der Web-Seite erscheint dann nur eine trockene Fehlermeldung, dass das Dokument nicht gültig (valide) sei.

Zum Redaktionszeitpunkt dieses Dokuments ist die HTML-Version 4.01 aktuell. Bei dieser HTML-Version und auch schon bei der glatten Versionsnummer 4.0 gibt es drei Sprachvarianten. Für eine davon müssen Sie sich entscheiden und den entsprechenden Dokumenttyp angeben.

### **Die Variante »Strict« für HTML**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

Benutzen Sie diese Angabe, wenn Sie bestimmte Elemente und Attribute nicht mehr verwenden wollen, die früher mal zum HTML-Standard gehörten, aber mittlerweile durch andere Möglichkeiten (vor allem durch *CSS Stylesheets*, siehe Kapitel 15) ersetzbar sind und deshalb nicht mehr zum »reinen« HTML gehören. Ferner sind die Verschachtelungsregeln für HTML-Elemente in



der Strict-Variante enger und im Sinne strukturierter Inhalte sauberer formuliert. So ist es in dieser Variante beispielsweise nicht erlaubt, zwischen `<body>` und `</body>` einfach nur Text zu notieren. Alle Inhalte müssen in so genannten Block-Elementen stehen, z.B. in Überschriften, Textabsätzen, Tabellen usw.

### Die Variante »Transitional« für HTML

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

Benutzen Sie diese Angabe, wenn Sie auch früher erlaubte Elemente und Attribute noch verwenden wollen. Ein Grund, sich für diese Variante zu entscheiden, könnte sein, dass Sie Ihr HTML so schreiben wollen, dass es auch mit Netscape 3.x und anderen älteren Browsern, die noch kein CSS verstehen, in etwa so aussieht, wie Sie es wünschen. Denn in der Strict-Variante entfallen praktisch alle Elemente und Attribute, die in HTML früher für das »optische Aufpäppeln« von Web-Seiten gesorgt haben. Dazu gehören Angaben zur Textausrichtung, zu Hintergrundfarben usw. All das können Sie auch mit CSS definieren, aber ältere Browser beherrschen noch kein CSS. In der Variante »Transitional« sind auch die Regeln für die Elementverschachtelung etwas milder. So ist es nach dieser Variante erlaubt, zwischen `<body>` und `<body>` »nackten Text« außerhalb eines weiteren Elements zu notieren.

### Die Variante »Frameset« für HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
  "http://www.w3.org/TR/html4/frameset.dtd">
```

Diese Angabe ist nur für spezielle HTML-Dateien gedacht, in denen Framesets definiert werden. Siehe dazu den Abschnitt 9.1, *Framesets und Frames definieren*.

### Dokumenttyp-Angaben für XHTML

Auch für XHTML 1.0 gibt es die drei Varianten »Strict«, »Transitional« und »Frameset«. Im Mittelteil der Dokumenttyp-Angabe muss jedoch bei Version 1.0 von XHTML XHTML 1.0 notiert werden. Auch die Web-Adressen für den Bezug sind andere als bei HTML.

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

**Dokumenttyp-Angaben für XHTML 1.1:**

Es gibt auch eine Sprachversion 1.1 von XHTML, die intern anders organisiert ist als XHTML 1.0. In XHTML 1.1 gibt es keine Sprachvarianten mehr. XHTML 1.1 entspricht nur noch der Sprachvariante »Strict« von XHTML 1.0.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

**Ältere Dokumenttyp-Angaben**

In manchen Ausnahmefällen kann es sinnvoll sein, sich auch auf eine ältere HTML-Version zu beziehen. Tun Sie dies jedoch nur, wenn irgendwelche technischen Gründe es erfordern. Folgende ältere Angaben gibt es:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 2.0//EN">
```

Benutzen Sie diese Dokumenttyp-Angabe, wenn Sie sich auf HTML 2.0 beziehen wollen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
```

Benutzen Sie diese Dokumenttyp-Angabe, wenn Sie sich auf HTML 3.2 beziehen wollen. Dokumenttyp-Angaben anderer HTML-Versionen sind nicht mehr im Gebrauch.

**3.2.4 Hinweis für die Praxis**

Wenn Sie nun entmutigt sind durch das ganze Wirrwarr von HTML und XHTML, HTML-Sprachvarianten, XML-Deklarationen und Dokumenttyp-Anweisungen – lassen Sie sich davon nicht zu sehr beeindrucken. Das Wirrwarr ist dadurch entstanden, dass es viele Entwicklungen gibt auf dem Gebiet der Auszeichnungssprachen und dass bei dieser Entwicklung auch nur mit Wasser gekocht wird. Vieles war zunächst unausgereift. Heute ist man bemüht, einen Spagat zu machen zwischen den fröhlichen Anfängen von HTML und den wachsenden Anforderungen, die professionelle Datenverarbeitung an HTML stellt.

Benutzen Sie für Ihre ersten Schritte in HTML zunächst einfach mal das oben vorgestellte Grundgerüst einer HTML-Datei, so wie es dort notiert ist. Lernen Sie den Umgang mit weiteren HTML-Elementen und ihren Attributen sowie den Umgang mit CSS Stylesheets kennen. Wenn Sie erst einmal vertrauter sind mit diesen Sprachen, werden Sie auch besser verstehen, welchen Sinn und Zweck die Unterschiede zwischen HTML und XHTML und die Unterschiede zwischen Sprachvarianten wie »Strict«, »Transitional« und »Frameset« haben.

**3.3 Regeln beim Editieren von HTML**

Wenn Sie einen Texteditor oder einen quelltextorientierten HTML-Editor zum Bearbeiten Ihrer HTML-Dateien verwenden, sollten Sie folgende Regeln kennen und beachten:

- Notieren Sie in einer neuen Datei immer zuerst das Grundgerüst (siehe Kapitel 3.2.2) einer HTML-Datei.

- Beachten Sie bei der Texteingabe die Maskierungsvorschriften für Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen (Kapitel 3.4).
- Setzen Sie Zeilenumbrüche und Leerzeilen so, dass Sie im Quelltext eine optimale Übersicht behalten. Beachten Sie aber auch, dass Zeilenumbrüche und Absatzschaltungen im Web-Browser nicht so angezeigt werden, wie sie im Quelltext eingegeben wurden. Für Zeilenumbrüche und Absatzschaltungen, die im WWW-Browser wirksam sein sollen, müssen Sie die entsprechenden HTML-Elemente verwenden, zum Beispiel diejenigen für Textabsätze (Absatzschaltungen) oder Zeilenumbrüche (Kapitel 5.3). Wenn Sie aus besonderen Gründen Text im Web-Browser so anzeigen wollen, wie Sie ihn eingeben (mit allen Einrückungen, Umbrüchen usw.), können Sie das HTML-Element für präformatierten Text (Kapitel 5.6) verwenden.
- Beachten Sie, dass es in HTML keine Tabulatoren gibt. Ein im Quelltext eingegebener Tabulator wird bei der Anzeige im Web-Browser in ein Leerzeichen umgewandelt. Zeilenumbruchzeichen, Tabulatorzeichen und einfache Leerzeichen bilden in HTML die Klasse der so genannten **White-Space-Zeichen** (*white space* = »weißer Raum«). Die Browser setzen in der Regel ein Tabulatorzeichen oder Zeilenumbruchzeichen im Editor als Leerzeichen im HTML-Text um. Mehrere solcher White-Space-Zeichen hintereinander werden ignoriert und zu einem einzigen Leerzeichen zusammengefasst. Um mehrere Leerzeichen hintereinander zu erzwingen, können Sie anstelle der normalen Leerzeicheneingabe die Zeichenfolge `&#160;` (geschütztes Leerzeichen) eingeben, und zwar so oft hintereinander wie gewünscht.

## 3.4 Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen

### 3.4.1 Allgemeines zu Zeichen in HTML

Wenn Sie Ihre Texte – sagen wir in deutscher Sprache – einfach in einen HTML-Editor eintippen, den Text mit Hilfe von HTML-Elementen strukturieren und sich das Ganze dann im Web-Browser anzeigen lassen, wird in der Regel der gesamte eingegebene Text korrekt angezeigt. Das klingt selbstverständlich – ist es aber nicht. In der HTML-Datei stehen nämlich nicht Ihre eingegebenen Buchstaben und Satzzeichen, sondern Byte für Byte numerische Werte wie 75, 168 oder 32. Der Browser versucht nun herauszubekommen, nach welchem **Zeichensatz** er diese numerischen Werte interpretieren soll. Ob beispielsweise nach einem westeuropäischen, einem kyrillischen oder einem arabischen Zeichensatz. Wenn Sie keinerlei Angaben zum verwendeten Zeichensatz machen, dann wird der Browser am Ende seiner Bemühungen einfach den Zeichensatz verwenden, der in seinen Einstellungen voreingestellt ist. In Ihrem Browser, der vermutlich eine englische oder deutschsprachige Benutzerführung hat, ist vermutlich der Zeichensatz für westeuropäische Sprachen eingestellt, der so genannte **Latin-1-Zeichensatz** (ISO 8859-1). Weil der HTML-Editor oder Texteditor, den Sie beim Eintippen benutzen, vermutlich ebenfalls nach diesem Zeichensatz speichert, klappt alles wunderbar. Wenn Ihre Web-Seiten aber im Web stehen, kann es passieren, dass Besucher aus Osteuropa, Asien usw. vorbeikommen, die ganz andere Zeichensätze in ihren Web-Browsern voreingestellt haben. Solche Besucher werden dann lauter Zeichen ihres eigenen, vertrauten Zeichensatzes sehen – aber es wird ein wirrer Zeichensalat sein, dem man beim besten Willen keinen Sinn entnehmen kann.

HTML bietet Ihnen deshalb die Möglichkeit an, dem Browser mitzuteilen, welchen Zeichensatz Sie meinen. Dann liegt es am Browser, die von Ihnen gemeinten Zeichen am Bildschirm beispielsweise eines Seitenbesuchers aus Fernost so anzuzeigen, wie Sie sie eingetippt haben. Es gibt folgende Möglichkeiten, dem Browser mitzuteilen, welchen Zeichensatz bzw. welches bestimmte Zeichen aus einem anderen als dem voreingestellten Zeichensatz Sie meinen:

- **global für eine Datei:** dazu gibt es die Möglichkeit, im Dateikopf einer HTML-Datei in einem so genannten Meta-Tag eine Angabe zum Default-Zeichensatz zu notieren. Eine solche Angabe ist sehr zu empfehlen, da Sie dem Browser damit auf jeden Fall die Entscheidung leichter machen, nach welchem Zeichensatz die Bytes der HTML-Datei zu interpretieren sind.
- **für einzelne Zeichen:** Das ist vor allem dann sinnvoll, wenn Sie eine globale Angabe zum Zeichensatz gemacht haben, im Text aber einzelne Zeichen verwenden wollen, die in dem angegebenen Zeichensatz nicht vorkommen. Dabei gibt es wiederum zwei Möglichkeiten: Entweder Sie verwenden eine spezielle numerische Notation. Für häufiger verwendete Sonderzeichen stellt HTML aber auch so genannte **benannte Zeichen** zur Verfügung. Für beide Möglichkeiten sollten Sie sich einmal mit der HTML-Zeichenreferenz beschäftigen.

Falls Sie das Thema mit den numerischen Bytes und den Zeichensätzen vertiefen möchten, können Sie den Abschnitt *Computer und geschriebene Sprache* (Kapitel 23.1) lesen.

### 3.4.2 Deutsche Umlaute und scharfes S

Wenn Sie in Ihrer HTML-Datei sonst keine Angaben zum verwendeten Zeichensatz machen, sollten Sie im Hinblick auf das Internet und die internationale Verwendung deutsche Umlaute und scharfes S durch spezielle, dafür vorgesehene **benannte Zeichen** ersetzen. Das gilt für den gesamten Inhalt einer HTML-Datei.

Ersetzen Sie das Zeichen ä durch die Zeichenfolge &auml ;  
 Ersetzen Sie das Zeichen Ä durch die Zeichenfolge &Auml ;  
 Ersetzen Sie das Zeichen ö durch die Zeichenfolge &ouml ;  
 Ersetzen Sie das Zeichen Ö durch die Zeichenfolge &Ouml ;  
 Ersetzen Sie das Zeichen ü durch die Zeichenfolge &uuml ;  
 Ersetzen Sie das Zeichen Ü durch die Zeichenfolge &Uuml ;  
 Ersetzen Sie das Zeichen ß durch die Zeichenfolge &szlig ;

Ein Beispiel:

```
In M&uuml;nchen steht ein Hofbr&auml;uhaus.  
Dort gibt es Bier aus Ma&szlig;kr&uuml;gen.
```

Ersetzen Sie die deutschen Sonderzeichen wie im obigen Beispiel durch die entsprechenden Zeichenfolgen – keine Sorge, die Web-Browser verstehen das und zeigen die Zeichen korrekt an.

**Beachten Sie:** Es gibt eine Menge weiterer Sonderzeichen, die durch solche Umschreibungen dargestellt werden sollten. Diese finden Sie in der HTML-Zeichenreferenz im Anhang aufgelistet.

### 3.4.3 Das Euro-Zeichen in HTML

Auch für das Euro-Zeichen gibt es ein **benanntes Zeichen** in HTML. Notieren Sie an der Stelle, wo das Euro-Zeichen stehen soll, die Zeichenfolge `&euro;`. Ein Beispiel:

```
Preis: &euro; 199,-
```

**Beachten Sie:** Vermeiden Sie unter MS Windows, das Euro-Zeichen über die Tastatur mit `[AltGr]+[e]` in HTML einzutippen. Denn Microsoft hat das Euro-Zeichen intern auf den Zeichenwert 128 gelegt hat, um es über Tastatur zugänglich zu machen und in vorhandene Schriftarten einzubauen. Das entspricht jedoch nicht dem Unicode/ISO-10646-Standard, auf dem HTML aufsetzt. Verwenden Sie in HTML deshalb das oben beschriebene `&euro;` oder als Alternative eine numerische Notation nach dem Unicode/ISO-10646-Standard. Dort hat das Euro-Zeichen den Hexadezimalwert 20AC oder den Dezimalwert 8364. Nach HTML 4.0 können Sie das Euro-Zeichen demnach numerisch so referenzieren: `&#x20AC;` oder `&#8364;`.

### 3.4.4 HTML-eigene Zeichen maskieren

Wenn in Ihrem Text Zeichen vorkommen, die in HTML eine bestimmte Bedeutung haben, müssen Sie diese Zeichen maskieren. Die folgenden Zeichen müssen Sie wie folgt maskieren:

Ersetzen Sie das Zeichen `<` durch die Zeichenfolge `&lt;`;  
 Ersetzen Sie das Zeichen `>` durch die Zeichenfolge `&gt;`;  
 Ersetzen Sie das Zeichen `&` durch die Zeichenfolge `&amp;`;  
 Ersetzen Sie das Zeichen `"` durch die Zeichenfolge `&quot;`;

Ein Beispiel:

```
Das ist ein &lt;HTML-Tag&gt;  
GmbH &amp; Co. KG  
&quot;Text steht in Anf&uuml;hrungszeichen&quot;
```

**Beachten Sie:** Am gefährlichsten ist die spitze öffnende Klammer (`<`). Wenn Sie dieses Zeichen nicht wie vorgeschrieben maskieren, bringen Sie den Web-Browser mit ziemlicher Sicherheit durcheinander, weil er glaubt, nun würde ein HTML-Tag folgen. Die anderen drei zu maskierenden Zeichen führen zwar meistens nicht zu Anzeigefehlern, doch sollten Sie sie auch stets maskieren. Besonders bei normalen Anführungszeichen, die ja doch sehr oft zum Einsatz kommen, wird die Maskierung im Text oft vergessen.

## 3.5 Farben definieren in HTML

Farben können Sie in HTML in vielen Zusammenhängen definieren, zum Beispiel bei dateiweiten Hinter- und Vordergrundfarben, bei Schriftfarben für Textabschnitte oder bei Hintergrundfarben in Tabellen. Allerdings sind all diese Angaben in HTML vom W3-Konsortium mittlerweile als *deprecated* (missbilligt) gekennzeichnet, d.h., sie sollen künftig nicht mehr zum Sprachstandard gehören. Der Grund ist, dass sich all diese Farben auch mit Hilfe von CSS Stylesheets definieren lassen. Denn CSS ist die Sprache für die Optik, nicht mehr HTML. Allerdings lohnt es sich trotzdem, sich mit dem Definieren von Farben in HTML zu befassen, weil in

CSS die gleichen Farbangaben möglich sind, dort aber darüber hinaus noch weitere Möglichkeiten bestehen. Im Abschnitt über *Farbangaben in CSS* wird das Thema aus CSS-Sicht behandelt (Kapitel 15.5).

Grundsätzlich gibt es zwei Möglichkeiten, Farben in HTML zu definieren:

- durch Angabe der RGB-Werte der gewünschten Farbe in Hexadezimalform (RGB = Rot/Grün/Blau-Wert der Farbe),
- durch Angabe eines Farbnamens.

Wenn Sie hexadezimale Werte angeben, arbeiten Sie Browser-unabhängig, und Sie haben die volle Auswahl unter 16,7 Millionen Farben. Wenn Sie Farbnamen angeben, umgehen Sie die etwas schwierige Definition einer Farbe im Hexadezimalmodus. Derzeit sind jedoch nur 16 Farbnamen offiziell standardisiert. Weitere Farbnamen gibt es, sie sind jedoch Browser-abhängig.

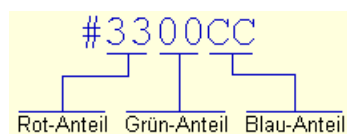
### 3.5.1 Hexadezimale Angabe von Farben

Wenn Sie Farben direkt im Hexadezimalmodus definieren, müssen Sie die gewünschte Farbe aus Angaben zu den drei Grundfarben Rot, Grün und Blau (RGB-Werte) zusammenstellen. Ein paar Beispiele:

```
<body bgcolor="#808080">
<table bgcolor="#00C0C0">
<tr color="#CC00CC">
```

Die erste Zeile erzeugt einen dunkelgrauen Hintergrund für die ganze Web-Seite, die zweite blaugrünen Hintergrund für eine Tabelle. Die letzte Zeile schließlich zeigt eine violette Trennlinie an.

Jede hexadezimale Farbdefinition ist 6-stellig und hat das Schema: #XXXXXX. Zunächst notieren Sie also ein Gatter #. Dahinter folgen sechs Stellen für die Farbdefinition. Die ersten beiden Stellen stellen den Rot-Wert der Farbe dar, die zweiten beiden Stellen den Grün-Wert, und die letzten beiden Stellen den Blau-Wert.



3.1: Die Farbangaben setzen sich aus drei zweistelligen Hexadezimalzahlen zusammen.

Die folgende Tabelle zeigt hexadezimale Ziffern:

| Hexadezimale Zahl | Dezimale Zahl |
|-------------------|---------------|
| 0                 | 0             |
| 1                 | 1             |
| 2                 | 2             |
| 3                 | 3             |

| Hexadezimale Zahl | Dezimale Zahl |
|-------------------|---------------|
| 4                 | 4             |
| 5                 | 5             |
| 6                 | 6             |
| 7                 | 7             |
| 8                 | 8             |
| 9                 | 9             |
| A                 | 10            |
| B                 | 11            |
| C                 | 12            |
| D                 | 13            |
| E                 | 14            |
| F                 | 15            |

Eine hexadezimale Ziffer kann also 16 Zustände haben. Für jeden Farbwert (Rot, Grün, Blau) stehen zwei Ziffern zur Verfügung. Das macht  $16 \times 16 (= 256)$  mögliche Zustände pro Farbwert.

**Beachten Sie:** Es gibt 16 Grundfarben, die von jedem VGA-kompatiblen Bildschirm angezeigt werden können.

| Hexwert | HTML-Farbname | Farbe         |
|---------|---------------|---------------|
| #000000 | black         | schwarz       |
| #800000 | maroon        | rotbraun      |
| #008000 | green         | dunkelgrün    |
| #808000 | olive         | ocker         |
| #000080 | navy          | dunkelblau    |
| #800080 | purple        | dunkelviolett |
| #008080 | teal          | blaugrün      |
| #C0C0C0 | silver        | hellgrau      |
| #808080 | gray          | dunkelgrau    |
| #FF0000 | red           | rot           |
| #00FF00 | lime          | hellgrün      |
| #FFFF00 | yellow        | gelb          |
| #0000FF | blue          | blau          |
| #FF00FF | fuchsia       | violett       |
| #00FFFF | aqua          | hellblau      |
| #FFFFFF | white         | weiß          |

Es gibt ferner 216 Standardfarben, die Netscape intern und plattformübergreifend zur Verfügung stellt. Diese Farbpalette hat sich im WWW zu einer Art Quasi-Standard entwickelt. Sie deckt das ganze Farbspektrum ab. Dabei sind für die einzelnen Rot-, Grün- und Blauwerte nur die Hexzahlen 00, 33, 66, 99, CC und FF zugelassen. So ergeben sich zum Beispiel Farbwerte wie #3300FF (hellblau), #CC99FF (blasses violett) oder #FF9900 (orange). Mehr über diese Paletten und ihre Bedeutung erfahren Sie im Kapitel 24.3.4 über Standardfarbpaletten.

### 3.5.2 Farbnamen für die 16 Grundfarben

Um eine Farbe mit Hilfe eines Farbnamens zu definieren, geben Sie anstelle des hexadezimalen RGB-Werts einfach den gewünschten Farbnamen an. Die folgenden Farbnamen sind Bestandteil von HTML seit Version 3.2 und werden von vielen WWW-Browsern verstanden.

```
<body bgcolor="black">
<table bgcolor="aqua">
<hr color="red">
```

Die drei Beispiele setzen einen schwarzen Seitenhintergrund, eine hellblaue Hintergrundfarbe für eine Tabelle und eine rote Trennlinie. Geben Sie den gewünschten Farbnamen an einer Stelle an, an der eine Farbangabe erlaubt ist. Um eine Übersicht über alle erlaubten Farben zu erhalten, rufen Sie das folgende Anzeigebeispiel auf. Dort sind auch die Hexadezimalwerte verzeichnet, die den Farbnamen entsprechen. Die Tabelle im vorhergehenden Abschnitt listet auch die Farbnamen auf.

### 3.5.3 Netscape-Farbnamen

Um eine Farbe mit Hilfe eines Farbnamens zu definieren, geben Sie anstelle des hexadezimalen RGB-Werts einfach den gewünschten Farbnamen an. Die spezielleren Farbnamen werden von Netscape und dem MS Internet Explorer interpretiert.

```
<body bgcolor="brown">
<table bgcolor="darkblue">
<hr color="pink">
```

Geben Sie den gewünschten Farbnamen an einer Stelle an, an der eine Farbangabe erlaubt ist. Eine Übersicht aller 120 erlaubten Farben bietet die folgende Tabelle. Dort sind auch die Hexadezimalwerte verzeichnet, die den Farbnamen entsprechen.

|                |         |
|----------------|---------|
| aliceblue      | #F0F8FF |
| antiquewhite   | #FAEBD7 |
| aquamarine     | #7FFFD4 |
| azure          | #F0FFFF |
| beige          | #F5F5DC |
| blueviolet     | #8A2BE2 |
| brown          | #A52A2A |
| burlywood      | #DEB887 |
| cadetblue      | #5F9EA0 |
| chartreuse     | #7FFF00 |
| chocolate      | #D2691E |
| coral          | #FF7F50 |
| cornflowerblue | #6495ED |
| cornsilk       | #FFF8DC |
| crimson        | #DC143C |
| darkblue       | #00008B |
| darkcyan       | #008B8B |
| darkgoldenrod  | #B8860B |



|                      |         |
|----------------------|---------|
| darkgray             | #A9A9A9 |
| darkgreen            | #006400 |
| darkkhaki            | #BDB76B |
| darkmagenta          | #8B008B |
| darkolivegreen       | #556B2F |
| darkorange           | #FF8C00 |
| darkorchid           | #9932CC |
| darkred              | #8B0000 |
| darksalmon           | #E9967A |
| darkseagreen         | #8FBC8F |
| darkslateblue        | #483D8B |
| darkslategray        | #2F4F4F |
| darkturquoise        | #00CED1 |
| darkviolet           | #9400D3 |
| deeppink             | #FF1493 |
| deepskyblue          | #00BFFF |
| dimgray              | #696969 |
| dodgerblue           | #1E90FF |
| firebrick            | #B22222 |
| floralwhite          | #FFFAF0 |
| forestgreen          | #228B22 |
| gainsboro            | #DCDCDC |
| ghostwhite           | #F8F8FF |
| gold                 | #FFD700 |
| goldenrod            | #DAA520 |
| greenyellow          | #ADFF2F |
| honeydew             | #F0FFF0 |
| hotpink              | #FF69B4 |
| indianred            | #CD5C5C |
| indigo               | #4B0082 |
| ivory                | #FFFFFF |
| khaki                | #F0E68C |
| lavender             | #E6E6FA |
| lavenderblush        | #FFF0F5 |
| lawngreen            | #7CFC00 |
| lemonchiffon         | #FFFACD |
| lightblue            | #ADD8E6 |
| lightcoral           | #F08080 |
| lightcyan            | #E0FFFF |
| lightgoldenrodyellow | #FAFAD2 |
| lightgreen           | #90EE90 |
| lightgrey            | #D3D3D3 |
| lightpink            | #FFB6C1 |
| lightsalmon          | #FFA07A |
| lightseagreen        | #20B2AA |
| lightskyblue         | #87CEFA |
| lightslategray       | #778899 |

|                   |         |
|-------------------|---------|
| lightsteelblue    | #B0C4DE |
| lightyellow       | #FFFFE0 |
| limegreen         | #32CD32 |
| linen             | #FAF0E6 |
| mediumaquamarine  | #66CDAA |
| mediumblue        | #0000CD |
| mediumorchid      | #BA55D3 |
| mediumpurple      | #9370DB |
| mediumseagreen    | #3CB371 |
| mediumslateblue   | #7B68EE |
| mediumspringgreen | #00FA9A |
| mediumturquoise   | #48D1CC |
| mediumvioletred   | #C71585 |
| midnightblue      | #191970 |
| mintcream         | #F5FFFA |
| mistyrose         | #FFE4E1 |
| moccasin          | #FFE4B5 |
| navajowhite       | #FFDEAD |
| oldlace           | #FDF5E6 |
| olivedrab         | #6B8E23 |
| orange            | #FFA500 |
| orangered         | #FF4500 |
| orchid            | #DA70D6 |
| palegoldenrod     | #EEE8AA |
| palegreen         | #98FB98 |
| paleturquoise     | #AFEEEE |
| palevioletred     | #DB7093 |
| papayawhip        | #FFefd5 |
| peachpuff         | #FFDAB9 |
| peru              | #CD853F |
| pink              | #FFC0CB |
| plum              | #DDA0DD |
| powderblue        | #B0E0E6 |
| rosybrown         | #BC8F8F |
| royalblue         | #4169E1 |
| saddlebrown       | #8B4513 |
| salmon            | #FA8072 |
| sandybrown        | #F4A460 |
| seagreen          | #2E8B57 |
| seashell          | #FFF5EE |
| sienna            | #A0522D |
| skyblue           | #87CEEB |
| slateblue         | #6A5ACD |
| slategray         | #708090 |
| snow              | #FFFAFA |
| springgreen       | #00FF7F |
| steelblue         | #4682B4 |

|             |         |
|-------------|---------|
| tan         | #D2B48C |
| thistle     | #D8BFD8 |
| tomato      | #FF6347 |
| turquoise   | #40E0D0 |
| violet      | #EE82EE |
| wheat       | #F5DEB3 |
| whitesmoke  | #F5F5F5 |
| yellowgreen | #9ACD32 |

## 3.6 Kommentare und Credits

### 3.6.1 Nicht angezeigter Text (Kommentar)

HTML bietet die Möglichkeit, an beliebigen Stellen innerhalb einer HTML-Datei Kommentare einzufügen. Kommentare werden von Web-Browsern ignoriert, d.h. bei der Präsentation nicht angezeigt. Kommentare sind z.B. sinnvoll, um interne Angaben zu Autor und Erstelldatum in einer Datei zu platzieren, um interne Anmerkungen zu bestimmten Textstellen zu machen oder um verwendete HTML-Befehle intern auszukommentieren. Ein Beispiel:

```
<h1>Willkommen!</h1>

<!-- Kommentar: das obendrüber ist eine Überschrift -->

<p>viel Text</p>

<!-- und das ist ein mehrzeiliger Kommentar
zu dem Text mit <p>...</p>
Letzte Zeile des Kommentars -->
```

Kommentare werden durch die Zeichenfolge `<!--` eingeleitet. Dahinter folgt beliebig langer Kommentartext. Innerhalb des Kommentartextes können Sie auch HTML-Elemente notieren. Alles, was zwischen der einleitenden Zeichenfolge und der beendenden Zeichenfolge `-->` steht, wird bei der Anzeige im Browser unterdrückt.

**Beachten Sie:** Vor allem beim Einbinden von JavaScript wird empfohlen, den Script-Code in HTML-Kommentare zu setzen, da einige ältere Browser darüber nicht hinweglesen, sondern den enthaltenen Text/Programmcode am Bildschirm ausgeben.

Während Sie an Ihren HTML-Dateien basteln, werden Sie häufig Verschiedenes testen und ausprobieren. Dabei ist es hilfreich, bisherige Lösungen vor dem Ausprobieren einer neuen Lösung nicht einfach zu überschreiben, sondern in einen Kommentar zu setzen. So müssen Sie sich nicht ärgern, wenn Sie etwas überschreiben, das eigentlich doch besser war als das, womit Sie es überschrieben haben.

### 3.6.2 Credits

Folgende Hinweise sollten Sie beachten, wenn Sie beabsichtigen, HTML-Dateien fürs Web zu erstellen:

- **Weisen Sie sich als Verfasser aus:**  
Weisen Sie darauf hin, wer die Web-Seiten erstellt hat. Anwender haben ein Recht zu erfahren, wer für eine Web-Seite verantwortlich ist.
- **Weisen Sie das Erstelldatum aus:**  
Schreiben Sie das Erstelldatum an eine geeignete Stelle im Text. Besonders bei schnell veraltenden Informationen ist dies wichtig. Eine Preisliste für PC-Hardware etwa, die seit zwei Jahren nicht mehr aktualisiert wurde, wird ihren Nutzen für heutige Anwender dann bereits im Erstelldatum verraten.
- **Weisen Sie Autorenrechte aus:**  
Weisen Sie auf Ihre Urheberrechte hin bzw. nennen Sie ausdrücklich die Bedingungen, was mit Ihrem Text geschehen darf und was nicht. Das Kopieren des Textes können Sie kaum verbieten, wohl aber die Weitergabe des Textes nach darin vorgenommenen Änderungen. Auch für Grafiken, z.B. Logos, sollten Sie entsprechende Angaben machen.
- **Bieten Sie die Möglichkeit zum Feedback an:**  
Dazu können Sie zum Beispiel einen E-Mail-Verweis (Kapitel 6.4) oder ein Formular (Kapitel 10) verwenden.

## 3.7 Referenzieren in HTML

HTML-Dateien bestehen bekanntlich nur aus Text. Dennoch enthalten viele Web-Seiten Grafiken, Hintergrundgrafiken, Multimedia-Elemente, Java-Applets, Flash-Animationen und dergleichen. Solche Elemente werden in HTML in Form einer Referenz auf eine entsprechende Datenquelle notiert. Auch ein ausführbarer Verweis zu einer eigenen oder fremden Web-Seite ist nur ausführbar, wenn er sein Verweisziel benennt. Für all diese Zwecke wird das Referenzieren in HTML benötigt. Ebenso gibt es in Ergänzungssprachen wie CSS Stylesheets oder JavaScript Stellen, an denen Sie andere Datenquellen referenzieren müssen.

Die Regeln zum Referenzieren sind dabei immer die gleichen. Der Grund ist das zentrale und einheitliche Adressierungs-Schema im Web, das unabhängig von der Syntax einzelner Betriebssysteme gilt und die genaue Adressierung beliebiger Quellen im Web erlaubt.

### 3.7.1 Mit vollständigen URIs referenzieren

Mit vollständigen URIs müssen Sie dann referenzieren, wenn sich die gewünschte Datenquelle grob gesagt nicht im aktuellen eigenen Web-Angebot befindet. Ein **URI** (*Universal Resource Identifier = universelle Quellenbezeichnung*) ist beispielsweise so etwas wie

`http://www.teamone.de/` oder

`http://selfhtml.teamone.de/html/allgemein/referenzieren.htm`

Beide Beispieladressen sind aber gleichzeitig auch so genannte **URLs** (*Uniform Resource Locators – einheitliche Quellenorter*). Und dann gibt es – um die Verwirrung komplett zu machen – auch noch so genannte **URNs** (*Uniform Resource Names – einheitliche Quellennamen*). Letztere sind dazu gedacht, um nicht wirklich existierende Datenquellen oder Quellen, die zwar existieren, aber durch kein bekanntes Internet-Protokoll im Netz abrufbar sind, dennoch eindeutig zu benennen. Ein URI ist also der Oberbegriff für URL und URN, wobei URI und URL bei typischen Adressen, hinter denen sich konkrete Dateien oder Datenquellen verbergen, faktisch das Gleiche sind. Im HTML-Standard wird aber konsequent von URIs geredet. Beispiele für URIs:

```
http://www.ihr-guter-name.de/
http://www.ihr-guter-name.de/index.htm
http://www.ihr-guter-name.de/index.htm#impressum
http://www.ihr-guter-name.de/hintergrund.gif
http://www.ihr-guter-name.de/presentation.pdf
http://www.ihr-guter-name.de/cgi-bin/suche.cgi?ausdruck=Hasenjagd
http://www.google.com/search?hl=de&safe=off&q=Stefan+M%FCnz&lrm=
ftp://www.ihr-guter-name.de/presentation.pdf
http://192.168.78.10/
http://www.ihr-guter-name.de:8082/geheim.htm
```

Ein vollständiger URI besteht aus der Angabe eines Internet-Protokolls, z.B. `http` oder `ftp`, gefolgt von einem Doppelpunkt. Dahinter kann – das ist von Protokoll zu Protokoll verschieden – eine Zusatzangabe zu einem lokalen Netzwerknamen möglich sein. Diese Angabe wird in zwei Schrägstriche `//` eingeschlossen. Bei den meisten Adressen gibt es keine solche Angabe, weshalb die beiden Schrägstriche dort einfach ohne Inhalt nebeneinander stehen.

Hinter diesen Angaben folgt die Adresse des Host-Rechners im Netz, auf dem sich die Datenquelle befindet. Das kann ein Domain-Name oder eine numerische IP-Adresse sein. Der Domain-Name ist übrigens in den obigen Beispielen `ihr-guter-name.de`. Das `www` davor ist eine im Web typische und bei Web-Servern einstellbare Voreinstellung für Sub-Domains.

Hinter der Adressierung des Host-Rechners kann – durch einen Doppelpunkt abgetrennt, eine so genannte Portnummer folgen, wie im letzten der obigen Beispiele bei `:8082`. Das ist immer dann erforderlich, wenn die Datenquelle nicht über den Standard-Port des angegebenen Protokolls wie etwa `http` erreichbar ist, sondern über einen anderen Port. In der Praxis benötigen Sie die Portangabe eher selten, aber kennen sollten Sie sie.

Dahinter folgt schließlich die lokale Pfadangabe zur gewünschten Datenquelle. Egal, um welches Betriebssystem es sich dabei handelt – Verzeichnispfade werden stets durch einfache Schrägstriche getrennt. Es ist Aufgabe der Server-Software auf dem Rechner, die Pfadangaben korrekt aufzulösen. Auf diese Weise brauchen Sie sich keine Gedanken zu machen, welches System der angesprochene Rechner benutzt.

Auf dem Rechner können beliebige Dateien und Datenquellen angesprochen werden. Voraussetzung ist, dass sie über das angegebene Protokoll wie z.B. `http` unter der Adressierung erreichbar sind. Es muss sich nicht unbedingt um Dateien handeln. So kann mit `#` und einem Namen dahinter etwa ein bestimmter Zielanker in einer HTML-Datei angesprochen werden. Wie solche Zielanker definiert werden, ist im Abschnitt *Anker definieren und Verweise zu Ankern* (Kapitel 6.2) beschrieben. Auch Aufrufe von CGI-Scripts mit Parametern sind URIs, wie im obigen Beispiel `suche.cgi?ausdruck=Hasenjagd`.

Zeichen, die nicht zum ASCII-Zeichensatz (Kapitel 23) gehören oder in URIs Bedeutung haben (z.B. der Schrägstrich, der Doppelpunkt oder das Prozentzeichen), müssen Sie innerhalb von URIs maskieren. Das geschieht durch Angabe eines Prozentzeichens % mit anschließendem Hexadezimalwert für das Zeichen. Im obigen Beispiel sehen Sie das z.B. bei M%FCnz, wobei FC die hexadezimale Angabe der Zahl 252 ist und diese wiederum den Buchstaben »ü« bedeutet.

**Beachten Sie:** Die voranstehenden Beschreibungen richten sich an Praktiker und erheben keinen Anspruch auf Vollständigkeit. Das Konzept der URIs ist noch wesentlich differenzierter als hier dargestellt. Bei Interesse können Sie es in der RFC 1630 ([www.ietf.org/rfc/rfc1630.txt](http://www.ietf.org/rfc/rfc1630.txt)) nachlesen.

Datenquellen im eigenen Web-Angebot können Sie natürlich auch mit vollständigen URIs referenzieren. Damit schränken Sie sich jedoch ein (siehe die einleitenden Bemerkungen zum Abschnitt 3.7.3 *Pfadangaben relativ zum Basis-URI*).

### 3.7.2 Absolute Pfadangaben relativ zum Basis-URI

Die Variante mit absoluten Pfadangaben wählen Sie, wenn die gewünschte Datenquelle auf dem gleichen Host-Rechner liegt und über das aktuelle Protokoll und den Standard-Port erreichbar ist. Das klingt komplizierter als es ist. In dem vollständigen URI <http://selfhtml.teamone.de/html/allgemein/referenzieren.htm> ist der Teil [/html/allgemein/referenzieren.htm](http://selfhtml.teamone.de/html/allgemein/referenzieren.htm) eine absolute Pfadangabe relativ zur Basis-URI <http://selfhtml.teamone.de>. Innerhalb des eigenen Web-Angebots und der eigenen Domain oder Sub-Domain können Sie also mit solchen Pfadangaben arbeiten. Einige Beispiele:

```
/
/index.htm
/index.htm#impressum
/hintergrund.gif
/praesentation.pdf
/cgi-bin/suche.cgi?ausdruck=Hasenjagd
/search?hl=de&safe=off&q=Stefan+M%FCnz&lr=
```

Der erste Schrägstrich hinter dem Basis-URI steht für das Wurzelverzeichnis des jeweiligen Internet-Servers. Es handelt sich meistens nicht um das tatsächliche Wurzelverzeichnis des Rechners oder der Festplatte, auf die Sie damit zugreifen. Bei Web-Servern ist beispielsweise einstellbar, welches tatsächliche Verzeichnis dem Web-Wurzelverzeichnis entsprechen soll. Hinter dem Basis-URI kann alles folgen, was schon im Abschnitt *Mit vollständigen URIs referenzieren* angesprochen wurde.

### 3.7.3 Pfadangaben relativ zum Basis-URI

Die relativen Pfadangaben wählen Sie, wenn Sie den jeweils aktuellen URI als Bezugs-URI nutzen. Dann können Sie von dort aus relativ adressieren. Die HTML-Datei mit dem URI <http://selfhtml.teamone.de/html/allgemein/referenzieren.htm> enthält beispielsweise zum Referenzieren einer Grafik folgende Angabe: `../src/logo.gif`. Das bedeutet: Gehe zwei Verzeichnisse nach oben, von dort aus ins Unterverzeichnis `src` und dort findest du die Datei `logo.gif`. Absolut gesehen, hat diese Datei also den URI <http://selfhtml.teamone.de/src/logo.gif>.

Diese Form der relativen Adressierung ist innerhalb von Web-Projekten sehr zu empfehlen. Der Grund ist, dass Sie das Web-Projekt auf diese Weise problemlos an eine andere Web-Adresse verschieben können, und trotzdem funktionieren noch alle projektinternen Verweise und Grafikreferenzen. Auch auf der lokalen Festplatte funktionieren dann alle Referenzen. Gerade wenn Sie Ihr Projekt auch mal auf CD-ROM oder anderen Medien veröffentlichen wollen, ist die relative Adressierung ein Muss. Ein paar Beispiele:

```
./
farben.htm
./farben.htm
bilder/grafik.gif
./bilder/grafik.gif
../
../../../../woanders/datei.htm
```

Eine Datei im gleichen Verzeichnis wie dem aktuellen können Sie einfach durch Angabe des Dateinamens referenzieren – im obigen Beispiel etwa die Datei `farben.htm`. Das aktuelle Verzeichnis referenzieren Sie durch `./` – also einen Punkt, gefolgt von einem Schrägstrich. Die Adressierung von `farben.htm` und `./farben.htm` im obigen Beispiel hat also den gleichen Effekt.

Eine Angabe wie `bilder/grafik.gif` referenziert eine Datei namens `grafik.gif` im Verzeichnis `bilder`, das ein Unterverzeichnis des aktuellen Verzeichnisses ist. Die Notation `./bilder/grafik.gif` hat wieder den gleichen Effekt wie `bilder/grafik.gif`.

Mit `../` referenzieren Sie das Verzeichnis über dem aktuellen Verzeichnis, egal wie es heißt. Mit `../../../../` referenzieren Sie das Verzeichnis über dem Verzeichnis über dem aktuellen Verzeichnis usw. Von jedem der so adressierten Verzeichnisse können Sie wieder auf deren Unterverzeichnisse zugreifen, wie im letzten der obigen Beispiele gezeigt.

## 3.8 Konventionen für Dateinamen

### 3.8.1 Dateinamen im Hinblick auf Server-Rechner

Die meisten Server-Rechner im heutigen Web haben ein Unix- oder von Unix abgeleitetes Betriebssystem. Unix unterscheidet strikt zwischen Groß- und Kleinschreibung bei Dateinamen. Wer sein Web-Projekt beispielsweise in einer DOS-/Windows-Umgebung erstellt, wo dies keine Rolle spielt, kann bittere Erfahrungen machen, wenn plötzlich die lokal funktionierenden Verweise und Grafikreferenzen nach dem Hochladen auf den Server-Rechner nicht mehr funktionieren. Der Grund ist in solchen Fällen, dass beispielsweise in den HTML-Dateien kleingeschriebene Verweisziele oder Grafikdateinamen notiert wurden, während das DOS-/Windows-System großgeschriebene Dateinamen an den Server übertragen hat.

Wenn Sie mit Windows 3.x oder DOS 6.x und niedriger arbeiten, bleibt Ihnen in einem solchen Fall kaum eine andere Möglichkeit, als die Dateien auf dem Server-Rechner mit Hilfe von FTP-Befehlen so umzubenennen, wie sie in den Verweisen und Grafikreferenzen notiert sind. Wenn Sie mit einem System arbeiten, das Groß-/Kleinschreibung bei Dateinamen zwar für die Optik, aber nicht intern unterscheidet (z.B. Windows seit Version 95), ist es am sichersten, Sie erzeugen

Dateinamen, die nur aus Kleinbuchstaben bestehen, und in Verweisen und Grafikreferenzen notieren Sie ebenfalls alle Dateinamen in Kleinbuchstaben.

### 3.8.2 Dateinamen im Hinblick auf Kompatibilität

Wenn Sie Ihr Web-Projekt ausschließlich für den Einsatz im Web erstellen, müssen Sie nur darauf achten, dass die Dateien vom Server-Rechner akzeptiert werden. Bei modernen Unix-Systemen, wie sie die meisten Web-Server einsetzen, sind Dateinamen bis zu 256 Zeichen Länge erlaubt. Vermeiden Sie aber in jedem Fall deutsche Umlaute und scharfes S in den Dateinamen. Als Sonderzeichen ist in jedem Fall der Unterstrich »\_« erlaubt. Andere Satzzeichen sollten Sie vermeiden. Fragezeichen »?«, Stern-Zeichen »\*«, Gleichheitszeichen »=« dürfen Sie auf keinen Fall in Dateinamen verwenden. Auch auf Leerzeichen sollten Sie unbedingt verzichten.

Wenn Sie Ihr Projekt auch anderweitig vertreiben möchten, zum Beispiel als Download zum Offline-Lesen oder auf CD-ROM, sollten Sie die engen Grenzen von MS-DOS-kompatiblen Dateinamen kennen und in Ihre Entscheidung mit einbeziehen. Unter MS-DOS ist maximal 1 Punkt zur Trennung von Dokumentname und Dateiendung erlaubt. Vor dem Punkt dürfen acht Zeichen stehen, dahinter drei Zeichen (für die Dateiendung).

### 3.8.3 Dateiendungen

Hierbei gilt generell: Halten Sie unbedingt die üblichen Konventionen für Dateiendungen ein. Moderne Web-Browser erkennen HTML-Dateien zwar auch an ihrem Inhalt, aber bei Verwendung im Web sollten Sie immer die üblichen Dateiendungen verwenden.

- Gewöhnliche HTML-Dateien erhalten entweder die Endung *.html* oder *.htm*.
- HTML-Dateien mit Server Side Includes erhalten die Endung *.shtml* oder *.shtm*.
- HTML-Dateien mit PHP-Abschnitten erhalten die Endung *.php*.
- HTML-Dateien mit ASP-Abschnitten erhalten die Endung *.asp*.

### 3.8.4 Default-Dateinamen bei Web-Servern

Die meisten heutigen Web-Server sehen einen oder mehrere bestimmte Dateinamen als Datei für die »Einstiegsseite« vor. Meistens ist das der Name *index.html* oder *index.htm*, manchmal auch *welcome.html* bzw. *welcome.htm* oder *default.html* bzw. *default.htm*. Bei einigen Providern können Sie auch selbst einen beliebigen Dateinamen als Default-Dateinamen bestimmen. Und wenn Sie selbst Zugriff auf die Konfiguration des Web-Servers haben, können Sie die Default-Dateinamen dort einstellen.

Default-Dateinamen haben den Vorteil, dass die Web-Adresse keinen HTML-Dateinamen mehr braucht und kürzer wird. Viele Adressen lauten ja einfach *http://www.xy.com/* oder *http://www.xy.com/verzeichnis/* – ohne weitere Angabe einer bestimmten HTML-Datei. Dass beim Aufruf einer solchen Adresse dennoch eine bestimmte HTML-Datei geladen wird, liegt eben daran, dass es eine Datei mit einem Namen gibt, die der Web-Server als Default-Dateiname gespeichert hat.



Erkundigen Sie sich bei Ihrem Provider, ob er einen Default-Dateinamen für Verzeichnisse anbietet, und wenn ja, welchen. Erstellen Sie Ihr Projekt dann so, dass die Einstiegsseite diesen Dateinamen hat und untergeordnete Seiten auf diese Datei zurückverweisen.

## 3.9 Informationsverteilung und Dateiorganisation

### 3.9.1 Hypertext schreiben – Information verteilen

HTML steht für Hypertext Markup Language. Die Sprache ist ausgelegt für Hypertext-gerechte Informationsverteilung. Der Anwender soll zwischen Informationen navigieren, die ihn interessieren. Wenn Sie dieser Themenkomplex näher interessiert, sollten Sie in der Einführung das Unterkapitel 1.5 über Hypertext lesen.

Bei der Hypertext-Informationsverteilung ist es nötig, dass die angebotene Information auf kleine, in sich abgeschlossene Informationseinheiten verteilt wird, die untereinander auf eine nachvollziehbare und einsichtige Weise vernetzt sind.

Wenn Sie beispielsweise Ihren Fußballverein im WWW präsentieren wollen, gehört dazu die übliche Einstiegsseite mit Verweisen zu den Unterseiten. Die Unterseiten sollten jeweils an gleicher Stelle einen Rückverweis auf die Einstiegsseite enthalten. Je eine Unterseite könnten Sie in diesem Beispiel etwa für die Geschichte, für die Vereinsberichte, für die aktuellen Tabellenstände, und für die derzeitigen Spieler spendieren. Die Seite für Spieler könnte nochmals Verweise zu Seiten enthalten, auf denen jeweils ein einzelner der Spieler vorgestellt wird. Bei solchen Unter-Unter-Seiten ist ein Rückverweis zur nächsthöheren Ebene wie auch zur Einstiegsseite angebracht. Wichtig sind auch die Querverweise. Wenn ein Spieler bereits lange dabei ist, findet sich bestimmt auch ein Querverweis zu einem wichtigen Ereignis der Vereinsgeschichte. Bei den Erläuterungen zum aktuellen Tabellenstand sind wiederum Querverweise zu den besten Torhütern usw. denkbar.

So entsteht ein sinnvolles Netz aus Einzelinformationen. Zusätzliche Verlinkungen sollten die Navigationsmöglichkeiten ergänzen. So ist es in vielen Fällen sinnvoll, so genannte Guided Tours anzubieten, also eine Abfolge von Seiten, die der Anwender bequem durchblättern kann.

Egal, welche Art von Projekt Sie in Angriff nehmen: Immer sollten Sie den hier skizzierten Blick für die Strukturierung und Vernetzung der Information im Auge behalten.

Was hier als »Seite« oder »Unterseite« bezeichnet wird, sollte in einem Web-Projekt jeweils eine HTML-Datei sein. Schrecken Sie nicht vor der Anzahl der entstehenden Dateien zurück. Die saubere Strukturierung der Information sollte oberstes Gebot sein. Die Technik der Vernetzung zwischen Projektdateien mit HTML wird im Abschnitt 6.2 über Verweise beschrieben. Weitere Tipps für die Konzeption Ihres WWW-Projekts erhalten Sie im Kapitel 25.1 *Web-Projekte planen*.

### 3.9.2 Dateiorganisation

Bei jedem etwas größeren Projekt werden Sie schnell feststellen, wie viele HTML-, Grafik- und andere Dateien sich dabei ansammeln. Um den Überblick zu behalten, sollten Sie sich entweder

ein sinnvolles Dateinamenschema überlegen, oder Sie legen Unterverzeichnisse an, etwa für Dateien zu einem Themengebiet oder für Grafiken usw. Wenn Sie Ihr Projekt im Web veröffentlichen wollen, müssen Sie beim Einsatz von Unterverzeichnissen vorher klären, ob Sie auf dem Web-Server, wo das Projekt einmal abgelegt werden soll, die Berechtigung bzw. Möglichkeit haben, Unterverzeichnisse anzulegen. Denn Sie werden in diesem Fall auf dem Server-Rechner die gleiche Verzeichnisstruktur erstellen müssen, die Sie lokal beim Erstellen gewählt haben, damit alle Verweise und Referenzen funktionieren.

### 3.10 Guter HTML-Stil

Definitive Richtlinien für »richtiges« HTML gibt es nicht. Wohl aber gibt es Regeln, wie ein gültiges HTML-Dokument auszusehen hat. Auf jeden Fall ist es empfehlenswert, sich so weit wie möglich an den HTML-Sprachstandards des W3-Konsortiums zu orientieren. Mittlerweile hat das W3-Konsortium auch erkannt, dass seine Aufgabe nicht nur im Ausarbeiten technischer Spezifikationen besteht, sondern auch in deren Vermittlung an »normalsterbliche« Anwender, die kein abgeschlossenes Informatikstudium haben. Wenn Sie sich intensiv und dauerhaft mit dem Erstellen eigener Web-Seiten beschäftigen, sollten Sie sich deshalb regelmäßig mal auf den Web-Seiten des W3-Konsortiums ([www.w3.org](http://www.w3.org)) nach Neuigkeiten und nach aktuellen Trends und Richtlinien umsehen. Es gibt auch ein deutsches Büro des W3-Konsortiums ([www.w3c.de](http://www.w3c.de)), das wichtige Informationen des W3-Konsortiums in deutscher Sprache anbietet.



3.2: Das World Wide Web Consortium W3C hat auch eine deutsche Startseite mit aktuellen News.

Für HTML-Autoren bietet das W3-Konsortium ferner einen Service an, um erstellte Dateien auf syntaktische Korrektheit zu überprüfen: den so genannten Validator (*validator.w3.org*). Damit können Sie Ihre Seiten, wenn diese im Web über eine HTTP-Adresse erreichbar sind, überprüfen lassen.

Was die Frage nach »gutem HTML-Stil« betrifft, ist es eigentlich viel besser zu beschreiben, wie man es auf keinen Fall machen sollte. Dazu gibt es eine zwar schon etwas betagte, aber immer noch lesenswerte Anleitung im Web: Die *Goldenen Regeln für schlechtes HTML* von Stefan Karzauninkat (*www.karzauninkat.com/Goldhtml*).



- 3.3: Die »Goldenen Regeln« von Stefan Karzauninkat sind zwar betagt, aber (leider) immer noch aktuell.

### 3.10.1 Nicht für bestimmten Browser schreiben

Wenn Sie Dateien im Web platzieren wollen, sollten Sie zum Testen Ihrer Dateien mehr als einen Web-Browser verwenden. Verwenden Sie die weit verbreiteten Produkte wie Netscape und den MS Internet Explorer, wenn es irgendwie geht auch in mehreren Versionen, aber auch mal ein älteres Produkt wie etwa Mosaic. Wenn Sie die Möglichkeit haben, auf mehreren Plattformen

(MS Windows, Macintosh, Sun usw.) zu testen, machen Sie unbedingt Gebrauch davon. Und vielleicht schauen Sie sich Ihre Seiten auch mal mit einem Handheld-Computer an oder mit einer Brille, durch die Sie alles undeutlich sehen.

Nun kann Ihnen niemand verbieten, HTML-Dateien zu schreiben, die nur von Netscape oder nur vom MS Internet Explorer korrekt angezeigt werden. Aber wenn Sie solche Seiten ins Netz stellen, sollten Sie wissen, dass Sie damit letztlich Ihrem Image schaden.

### 3.10.2 Nicht für bestimmte Bildschirmauflösung schreiben

Das Erzwingen von festen Breiten durch Einsatz von verschachtelten Tabellen und »blinden Pixeln« ist mittlerweile so verbreitet im Web, dass manche Homepage-Autoren schon gar nicht mehr wissen, dass einfaches HTML normalen Fließtext erzeugt, der sich der Fenstergröße des Browsers anpasst. Nun macht es keinen Sinn, das Erzwingen von Mindestbreiten völlig zu verwerfen. Tabellen mit mehreren Spalten und entsprechendem Inhalt benötigen nun mal eine gewisse Breite, und HTML ist auch nicht WML (eine Sprache speziell zum Erstellen von Web-Seiten für Handy-Displays). Wenn Sie aber vor der Frage stehen, ob Sie Ihre Web-Seiten für eine 1024er oder eine 800er Bildschirmauflösung »optimieren« sollen, dann stehen Sie vor der falschen Frage. Denn viele Anwender surfen mit mehreren Instanzen des Web-Browsers und öffnen Verweise zu anderen Seiten gerne in neuen Browser-Fenstern. Diese Fenster werden oft nicht im Vollbildmodus angezeigt. Auch haben viele Anwender Programme offen, die grundsätzlich einen Teil des Bildschirms einnehmen und immer angezeigt werden – z.B. Instant-Messaging-Programme wie ICQ. All diese Anwender machen Ihnen also ohnehin nicht die Freude, das Browser-Fenster extra wegen Ihren Seiten auf maximale Bildschirmgröße zu bringen. Grübeln Sie daher nicht lange über anzunehmende Durchschnittswerte von Bildschirmgrößen. Ein paar hundert Pixel (also eine sehr unbestimmte Angabe) in der Breite können Sie voraussetzen, aber das ist auch alles.

Verwenden Sie bei Tabellen oder Frames tendenziell eher prozentuale Breiten- und Höhenangaben. Absolute Pixelangaben haben nur dort einen Sinn, wo beispielsweise die erste Spalte einer blinden Tabelle über einem farblich zweigeteilten Hintergrundbild liegen soll. Ansonsten sollten Sie sich nicht zu sehr auf solche Vorstellungen wie die versteifen, dass eine Grafik oder ein Absatz bei jedem Anwender genau 10,8 cm vom linken Rand entfernt beginnt. Wenn das bei Ihnen gut aussieht, bedeutet das noch lange nicht, dass es bei jemand anderem gut aussieht, und in einigen Fällen könnte es auch ausgesprochen schlecht aussehen.

### 3.10.3 HTML-Elemente nicht zweckentfremden

Besonders die logisch-semantischen Elemente von HTML sollten Sie nicht benutzen, um bestimmte Formatier-Effekte zu erzielen. So verwenden einige Leute z.B. das `blockquote`-Element, um Absätze einzurücken, nur weil die Mehrzahl der Browser den Text, der in `<blockquote> ... </blockquote>` eingeschlossen ist, als eigenen, eingerückten Absatz darstellen. Das `blockquote`-Element ist jedoch für Zitate gedacht und sollte auch nur dafür verwendet werden.

Noch häufiger wird mit Überschriften Schindluder getrieben. Überschriften sind nicht dazu da, um Text groß und fett zu machen, sondern dazu, logische Hierarchieverhältnisse zwischen Text-

abschnitten zu markieren. Wenn Sie Text auffällig formatieren wollen, dann benutzen Sie dazu CSS Stylesheets.

### 3.10.4 Ehrliche und aussagekräftige Verweistexte verwenden

Generell gilt: Das Ziel eines Verweises sollte das halten, was der Verweis verspricht. Das bedeutet beim Setzen des Verweises: Der Verweistext sollte weder zu viel noch zu wenig versprechen. Wenn Sie beispielsweise Information über ein Softwareprodukt anbieten, ohne es zum Download anzubieten, ist es unfair, dem Anwender mit einem Verweis auf die Information zu suggerieren, er könne das Produkt auch gleich downloaden.

Verweise können in HTML an jeder beliebigen Stelle im Text stehen. Wenn Sie jedoch einmal Text lesen, in dem jedes zweite Wort ein Verweis ist, werden Sie schnell merken, dass dies den Lesefluss ungemein stört. Der Grund dafür ist, dass Verweise immer gleich die Aufmerksamkeit auf sich ziehen und den Leser von seiner eigentlichen Aufgabe, dem geistigen Erfassen des im Text Gemeinten, ablenken. Um so wichtiger ist es, dass Verweise innerhalb des Fließtextes dem Anwender keine Rätsel aufgeben, sondern sofort erfassbar sind.

Verwenden Sie einen Verweis innerhalb des Fließtextes also nur dann, wenn der Verweistext sinnvoll ist. Formulieren Sie Sätze, in denen verweissensitiver Text vorkommt, so, dass der Verweistext aussagekräftig ist. Schreiben Sie z.B. nicht: »Für weitere Information klicken Sie *hier*«, sondern: »*Weitere Information* ist ebenfalls verfügbar«.

### 3.10.5 Grafiken richtig einsetzen

Wenn Sie nicht gerade eine virtuelle Kunstaussstellung in HTML erstellen, sollten Sie sich mit großen Grafikdateien zurückhalten. Bedenken Sie bei Dateien, die Sie fürs Web erstellen, dass viele Anwender einen volumenabhängigen Internet-Zugang haben, d.h. sie zahlen dafür, wie viel Daten sie in den Arbeitsspeicher ihres Rechners laden. Zwar erlauben die meisten Browser, das Laden von Grafiken auszuschalten, aber wenn der Anwender keinen Gebrauch von dieser Funktion macht, heißt das noch lange nicht, dass er bereit ist, ohne Ankündigung eine 1-MByte-Grafik zu laden.

Versuchen Sie es daher lieber mit kleinen, wohlplatzierten Grafiken. Oft genügen 16 Farben statt 256 oder gar 16,7 Mio. Das macht die Grafiken deutlich kleiner.

Andererseits sollten Sie keinesfalls auf den Einsatz von Grafiken verzichten. Reiner Text ist am Bildschirm nämlich wesentlich ermüdender zu lesen als in Printmedien. Deshalb sollten Sie längere Texte möglichst reichhaltig strukturieren und auflockern. Dazu gehört auch die Verwendung von Grafiken. Ideal sind kleine Grafiken in Icon-Größe. Die sind schnell geladen, und Sie können bedenkenlos mehrere davon pro HTML-Datei referenzieren. Kleine Grafiken können auch bestimmte Corporate-Identity-Funktionen oder Orientierungsfunktionen übernehmen. Der Vorteil solcher mehrfach verwendeter Grafiken ist, dass die meisten Browser sie nur einmal laden und dann im Speicher halten.

## 4 Dateiweite Einstellungen

### 4.1 HTML-Kopfdaten

#### 4.1.1 Titel einer HTML-Datei

Jede HTML-Datei sollte einen Titel erhalten. Das ist aus folgenden Gründen besonders wichtig:

- Der Titel der Datei wird bei der Anzeige im Web-Browser in der Titelzeile des Anzeigefensters angezeigt.
- Der Titel der Datei wird vom Web-Browser beim Setzen von Lesezeichen (Bookmarks, Favoriten) auf die Datei verwendet.
- Der Titel der Datei wird im Web-Browser in der Liste der bereits besuchten Seiten angezeigt.
- Der Titel der Datei dient im Web vielen automatischen Suchprogrammen als wichtiger Input. Wenn die Datei zu den Suchtreffern einer Suche gehört, bieten viele Suchmaschinen den Titel der Datei als anklickbaren Verweis an.

Ein Beispiel:

```
<head>
<title>Ausblick vom Hamburger Michel</title>
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Innerhalb des Grundgerüsts einer HTML-Datei wird der Titel im Dateikopf notiert. `<title>` leitet die Angabe des Titels ein. Dahinter folgt der Text des Titels. Mit `</title>` wird die Titelangabe abgeschlossen (*title* = *Titel*).

**Beachten Sie:** Der Titeltext sollte nicht zu lang sein. Für beschreibende Angaben zur Datei gibt es die Meta-Angaben zum Inhalt (Kapitel 4.2). Für den Titeltext gelten alle Regeln für Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen.

#### 4.1.2 Meta-Angaben zum Inhalt



#### *Allgemeines zu Meta-Angaben*

In Meta-Angaben können Sie verschiedene nützliche Anweisungen für Web-Server, Web-Browser und automatische Suchprogramme im Internet (»Robots«) notieren. Meta-Angaben können Angaben zum Autor und zum Inhalt der Datei enthalten. Sie können aber auch HTTP-Befehle absetzen, zum Beispiel zum automatischen Weiterleiten des Web-Browsers zu einer anderen Adresse. Meta-Angaben sind **nicht** für persönliche Anmerkungen oder Kommentare

zum Quelltext der HTML-Datei gedacht. Für solche Angaben können Sie Kommentare (Kapitel 3.6) definieren.

Der HTML-Standard seit Version 4.0 schreibt keine konkreten Meta-Angaben mehr vor, sondern definiert lediglich den grundsätzlichen Aufbau einer Meta-Angabe. Zur Standardisierung von Meta-Angaben arbeitet das W3-Konsortium stattdessen an einer Sprache namens Resource Description Framework (RDF). Auf den Web-Seiten des W3-Konsortiums finden Sie Informationen zum Resource Description Framework ([www.w3.org/RDF/](http://www.w3.org/RDF/)).

Für jede Meta-Angabe notieren Sie ein Meta-Tag im HTML-Dateikopf. Es ist also kein Problem, mehrere Meta-Tags zu notieren. Einige Beispiele:

```
<head>
<meta name="author" content="Anna Lyse">
<meta http-equiv="expires"
  content="Sat, 01 Dec 2001 00:00:00 GMT">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Eine Meta-Angabe steht in einem Standalone-Tag namens `meta`. Jedes Meta-Tag hat dann in der Regel zwei Attribute. Das eine Attribut lautet entweder `name=` oder `http-equiv=`, und das andere lautet stets `content=`. Durch die beiden Attribute lassen sich typische »Konfigurationsdaten« vom Typ »Eigenschaft=Wert« erzeugen. Im obigen Beispiel wird auf diese Weise z.B. mit `name=` eine Eigenschaft "author" angegeben, der mit `content=` ein Wert Anna Lyse zugewiesen wird. In der zweiten Meta-Angabe des Beispiels wird einer Eigenschaft namens `expires` der Wert Sat, 01 Dec 2001 00:00:00 GMT zugewiesen. In dem zweiten Beispiel wird die Eigenschaft jedoch mit `http-equiv=` definiert.

Der Unterschied zwischen `name=` und `http-equiv=` ist, dass `name=` allgemeine, nicht näher spezifizierte Eigenschaften benennen kann, während `http-equiv=` Eigenschaften benennt, die ein Web-Server auslesen sollte. Eigenschaften, die mit `name=` definiert werden, richten sich daher tendenziell eher an auslesende Client-Programme, also an Web-Browser, aber auch an Suchmaschinen-Robots, die Web-Seiten zum Füttern ihrer Suchmaschinendatenbank auslesen. Eigenschaften, die mit `http-equiv=` definiert werden, sind dagegen für den Web-Server gedacht. Die Idee dahinter ist, dass der Web-Server, wenn er von einem aufrufenden Browser die Anfrage erhält, diese HTML-Datei zu übermitteln, die Meta-Tags vorher ausliest und Angaben, die mit `http-equiv=` definiert wurden, in den HTTP-Header einbaut, den er an den aufrufenden Web-Browser schickt. Als Autor der HTML-Datei können Sie auf diese Weise also die Kommunikation zwischen Web-Server und Web-Browser beeinflussen. Inwieweit Angaben dieser Art etwas bewirken, hängt also vom Web-Server ab bzw. davon, ob er diese Daten vor dem Übertragen der Datei an den Browser ausliest oder nicht. Für Wertzuweisungen an `content=` gelten alle Regeln für Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen.

**Beachten Sie:** Mit Meta-Angaben wird leider viel Missbrauch getrieben. Da viele Meta-Angaben keine unmittelbar nachvollziehbare Wirkung am Bildschirm haben und andererseits mit Suchdiensten zu tun haben, regen sie offenbar magisch die Phantasie mancher Geister an, die meinen, mit einem Mammutaufgebot irgendwo aufgeschnappter Meta-Angaben würden sie über Nacht zum Superstar, da ihre Homepage dann immer ganz oben bei den Ergebnislisten der Suchmaschinen erscheint. Dazu nur so viel: Homepages werden nicht durch Meta-Angaben erfolgreich,

sondern durch Inhalt. Zu einer sauber gemachten Homepage gehört aber auch der sinnvolle Einsatz von Meta-Angaben.

### **Beschreibung, Autor, Stichwörter, Datum**

Die hier beschriebenen Meta-Angaben sind am verbreitetsten, sie werden von fast allen großen Suchmaschinen im Web erkannt und ausgelesen. Sie können für Suchprogramme Ihre Autorenschaft, eine Kurzbeschreibung des Inhalts, charakteristische Stichwörter und das Publikationsdatum notieren. Häufig werden diese Daten in der Suchmaschine angezeigt, wenn ein Anwender nach etwas sucht und diese Datei zu den Treffern gehört. Es ist also sinnvoll, diese Meta-Angaben in allen inhaltlich relevanten HTML-Dateien zu notieren und dabei redaktionell genauso sorgfältig zu sein wie bei dem Text, der im Browser-Fenster angezeigt wird. Ein Beispiel:

```
<head>
<meta name="description" content="Dieser
  Beschreibungstext soll einem Anwender im
  Suchdienst bei Auffinden dieser Datei erscheinen.">
<meta name="author" content="Rainer Wahnsinn">
<meta name="keywords" content="HTML, Meta-Informationen,
  Suchprogramme, HTTP-Protokoll">
<meta name="date" content="2001-12-15T08:49:37+00:00">
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

- Mit `<meta name="description" content="Beschreibungstext">` bestimmen Sie einen Beschreibungstext (*meta* = »über«, *name* = Name, *description* = Beschreibung, *content* = Inhalt).
- Mit `<meta name="author" content="Autorenname">` teilen Sie den Namen des Autors (d.h. des inhaltlich Verantwortlichen) für die HTML-Datei mit (*author* = Autor).
- Mit `<meta name="keywords" content="[Wortliste]">` bestimmen Sie Stichwörter für ein Suchprogramm. Ein Anwender, der in der Suchdatenbank des Suchprogramms nach einem dieser Stichwörter sucht, soll die aktuelle HTML-Datei bevorzugt als Suchtreffer angezeigt bekommen, da das Stichwort in dieser Datei ein zentrales Thema ist (*keywords* = Schlüsselwörter). Trennen Sie die einzelnen Schlüsselwörter durch Kommata. Ein Schlüsselwort kann durchaus aus zwei oder mehreren Einzelwörtern bestehen. Halbe oder ganze Sätze gehören hier allerdings nicht hin, denn dafür gibt es den Beschreibungstext, den Sie bei *name="description"* notieren können.
- Mit `<meta name="date" content="[Datum/Uhrzeit]">` geben Sie an, wann die Datei publiziert wurde. Das Schema ist dabei eine UTC-Zeitangabe. Im obigen Beispiel ist 2001 die Jahreszahl, 12 der Monat (Dezember), 15 der Tag, 08 die Stunden, 49 die Minuten und 37 die Sekunden. Die Angabe 00:00 hinter dem Pluszeichen ist die Abweichung der Zeit von der Greenwich-Zeit in Stunden und Minuten, im Beispiel keine Abweichung. Wenn Sie nur das Datum, aber keine Uhrzeit angeben wollen, notieren Sie nur den Teil der Datumsangabe bis vor dem großen T (für *Time*).



### Meta-Angaben nach Dublin-Core

Eine internationale Gruppe von Experten, unter dem Namen **Dublin Core** vereint, hat ein System für Meta-Angaben veröffentlicht. Das System ist einfach zu handhaben und berücksichtigt alle wichtigen Angaben, die zu einem Dokument gemacht werden können. Die Meta-Angaben von Dublin-Core werden auch vom W3-Konsortium begrüßt. Einer Verwendung steht also nichts im Wege. Im Web finden Sie Informationen zur Dublin Core Metadata Initiative ([dublincore.org/](http://dublincore.org/)).

Zumindest die führenden Suchmaschinen im Web erkennen mittlerweile die Meta-Angaben nach Dublin-Core. Das führt bei solchen Suchmaschinen zu einer sehr ordentlichen Suchtrefferanzeige, wenn diese Datei zu den Suchtreffern gehört. Ein Beispiel:

```
<head>
<meta name="DC.Title" content="SELFHTML/Meta-Angaben">
<meta name="DC.Creator" content="Stefan M&uuml;l;nz">
<meta name="DC.Subject" content="Meta-Angaben">
<meta name="DC.Description"
    content="Heute bekannte Meta-Angaben in HTML">
<meta name="DC.Publisher" content="TeamOne">
<meta name="DC.Contributor" content="Wolfgang Nefzger">
<meta name="DC.Date" content="2001-12-15">
<meta name="DC.Type" content="Text">
<meta name="DC.Format" content="text/html">
<meta name="DC.Identifier" content="http://selfhtml.teamone.de/">
<meta name="DC.Source" content="HTML-Referenz">
<meta name="DC.Language" content="de">
<meta name="DC.Relation" content="Unterkapitel">
<meta name="DC.Coverage" content="Munich">
<meta name="DC.Rights"
    content="Alle Rechte liegen beim Autor">
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Bei Meta-Angaben, die einem bestimmten öffentlichen System angehören, so wie hier dem System von Dublin Core, wird dem Namen einer Meta-Angabe eine Kurzbezeichnung für den Herausgeber der Meta-Angaben vorangestellt. Bei Dublin Core sind das die Initialen DC. Dahinter folgt, durch einen Punkt getrennt, der Name der Meta-Angabe. Der Zuweisungswert erfolgt wie üblich über das Attribut `content=`.

- Mit `<meta name="DC.Title" content="Titel">` können Sie einen Titel für die Datei angeben, ähnlich wie im `title`-Element von HTML (*title* = *Titel*).
- Mit `<meta name="DC.Creator" content="Name">` weisen Sie den geistigen Urheber des Inhalts der Datei aus (*Creator* = *Ersteller*).
- Mit `<meta name="DC.Subject" content="Text">` geben Sie das Thema an, das in der Datei behandelt wird (*Subject* = *Thema*).
- Mit `<meta name="DC.Description" content="Text">` können Sie eine Kurzbeschreibung des Dateiinhalts notieren (*Description* = *Beschreibung*).

- Mit `<meta name="DC.Publisher" content="Name">` können Sie angeben, wer für die Publikation der Datei verantwortlich ist. Das kann der Autor selbst sein, aber auch etwa ein Verlag oder der Nachfahre eines verstorbenen Autors (*Publisher* = *Veröffentlicher*).
- Mit `<meta name="DC.Contributor" content="Name">` können Sie Koautoren oder Personen angeben, die neben dem Hauptautor maßgeblich am Inhalt beteiligt sind, z.B. Grafiker, Musiker, Übersetzer (*Contributor* = *Beitragender*). Falls Sie mehrere Angaben dazu machen, trennen Sie die Namen durch Kommata.
- Mit `<meta name="DC.Date" content="Datum">` können Sie angeben, wann die Datei publiziert wurde. Im obigen Beispiel ist 2001 die Jahreszahl, 12 der Monat (Dezember) und 15 der Tag.
- Mit `<meta name="DC.Type" content="Typ">` können Sie die Textsorte bzw. den Dokumententyp angeben, dem sich der Inhalt der Datei zuordnen lässt. Die erlaubten Zuweisungen an `content=` fasst die folgende Tabelle zusammen:

| Dokumenttyp          | Beschreibung                                                                                                                                                   |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Collection           | Wenn die Datei ein Verzeichnis mit Verweisen zu Unterseiten ist.                                                                                               |
| Dataset              | Wenn die enthaltene Information datensatzartig vorliegt, z.B. in einer Tabelle.                                                                                |
| Event                | Wenn der Inhalt ein zeitliches Ereignis behandelt, z.B. eine Ausstellung oder eine Hochzeitsfeier.                                                             |
| Image                | Wenn der Inhalt primär eine grafische Darstellung ist oder ein zentrales Foto.                                                                                 |
| Interactive Resource | Wenn der Zweck der Seite in der Interaktion mit dem Anwender liegt, z.B. wenn ein Java-Applet eingebettet ist, das einen Chat realisiert.                      |
| Service              | Wenn der Inhalt eine Dienstleistung anbietet.                                                                                                                  |
| Software             | Wenn der Inhalt Software anbietet, z.B. zum Download.                                                                                                          |
| Sound                | Wenn der Inhalt primär akkustisch ist, z.B. ein eingebettetes MP3-Musikstück zum Abspielen anbietet oder einen laufenden Radiosender über RealAudio einbettet. |
| Text                 | Wenn der Inhalt textorientiert ist – egal, ob es sich um Sachtexte, Literatur, Witze, Nachrichten oder wissenschaftliche Abhandlungen handelt.                 |

- Mit `<meta name="DC.Format" content="Typ">` können Sie das Datenformat bzw. den Medientyp angeben, für den der Inhalt der Datei gedacht ist. Als Wert bei `content=` können Sie einen Mime-Type (Anhang A.2) angeben. Der Mime-Type für HTML lautet `text/html`.
- Mit `<meta name="DC.Identifier" content="Wert">` können Sie eine eindeutige Nummer oder Bezugsadresse für die Datei angeben (*Identifier* = *eindeutiger Bezeichner*). Bei Web-Seiten ist das beispielsweise der Original-URI.
- Mit `<meta name="DC.Source" content="Quelle">` können Sie eine Quelle nennen, von der die aktuelle Datei abgeleitet ist (*Source* = *Quelle*). Das kann beispielsweise der Fall sein, wenn eine HTML-Datei Auszüge aus einem Buch beinhaltet. In diesem Fall können Sie das Buch als Quelle angeben.

- Mit `<meta name="DC.Language" content="Sprache">` können Sie die Landessprache angeben, in der der Inhalt der Datei verfasst ist (*Language = Sprache*). Als Angabe bei `content=` wird ein Sprachenkürzel (Anhang A.3) erwartet.
- Mit `<meta name="DC.Relation" content="Quelle">` können Sie angeben, welche Beziehung zwischen der aktuellen Datei und einem zugehörigen Editionsprojekt bestehen (*Relation = Bezug*). Der Wert bei `content=` kann z.B. eine verbale Beschreibung der Bezugs-Art sein.
- Mit `<meta name="DC.Coverage" content="Text">` können Sie einen zeitlichen oder geografischen Bezugspunkt für den Inhalt der Datei angeben (*Coverage = Erfassung, Berichterstattung*). Das kann bei geografischen Angaben etwa ein Ort, eine Längen-/Breitengradangabe oder eine Region sein, bei zeitlichen Angaben ein Datum oder ein Zeitalter.
- Mit `<meta name="DC.Rights" content="Text">` können Sie eine Angabe zum Copyright des Dateiinhalts machen (*Rights = Rechte*). Sie können dabei aber auch einen URI angeben, unter dem die genauen Copyright-Angaben genannt werden.

**Beachten Sie:** Sie müssen nicht alle Meta-Angaben des Dublin-Core-Sets verwenden. Es ist kein Problem, wenn Sie nur zu denjenigen etwas notieren, wo Sie sinnvolle Angaben machen können.

### Auslesen erlauben/verbieten

Die folgenden Angaben werden von den meisten Suchmaschinen-Robots beachtet. Wenn Sie eine HTML-Datei zwar im Web anbieten, aber trotzdem verhindern möchten, dass die Datei über öffentliche Suchdienste auffindbar sein soll, können Sie eine entsprechende Anweisung als Meta-Information für Suchprogramme notieren. Sie können solchen Suchprogrammen aber auch mit einer entsprechenden Eingabe signalisieren, dass Sie ein Auslesen der Datei und aller Dateien, die darin über Verweise erreichbar sind, ausdrücklich wünschen. Ein Beispiel:

```
<head>
<meta name="robots" content="noindex">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit `<meta name="robots" content="noindex">` verbieten Sie einem Suchprogramm, Inhalte aus der HTML-Datei an seine Suchdatenbank zu übermitteln (*robots = Suchprogramme, content = Inhalt, noindex = keine Indizierung*). Anstelle von `noindex` können Sie auch `none` notieren (*none = keine*).

Daneben sind folgende weitere Angaben möglich:

- `<meta name="robots" content="index">`: Damit erlauben Sie einem Suchprogramm ausdrücklich, Inhalte aus der aktuellen HTML-Datei an seine Suchdatenbank zu übermitteln (*index = Indizierung*).
- `<meta name="robots" content="nofollow">`: Damit erlauben Sie einem Suchprogramm, Inhalte aus der aktuellen HTML-Datei an seine Suchdatenbank zu übermitteln (*nofollow = nicht folgen*). Sie verbieten dem Suchprogramm jedoch, untergeordnete Dateien Ihres Projekts, zu denen Verweise führen, zu besuchen.
- `<meta name="robots" content="follow">`: Damit erlauben Sie einem Suchprogramm ausdrücklich, Inhalte aus der aktuellen HTML-Datei und aus untergeordneten Dateien Ihres

Projekts, zu denen Verweise führen, zu besuchen und an seine Suchdatenbank zu übermitteln (*follow = folgen*).

**Beachten Sie:** Wenn Sie sich für das Erlauben und Verboten von Inhalten für Such-Robots interessieren, sollten Sie auch den Abschnitt Anhang A.6 *robots.txt – Robots kontrollieren* lesen.

### ***Sprache bei Stichwörtern***

Für Meta-Angaben zu Stichwörtern (bei solchen zur Kurzbeschreibung des Inhalts sollte es auch funktionieren) können Sie mehrere gleichartige Meta-Angaben notieren und dabei zwischen Landessprachen trennen. Ein Beispiel:

```
<head>
<meta name="keywords" lang="de"
  content="Ferien, Griechenland, Sonnenschein">
<meta name="keywords" lang="en-us"
  content="vacation, Greece, sunshine">
<meta name="keywords" lang="en"
  content="holiday, Greece, sunshine">
<meta name="keywords" lang="fr"
  content="vacances, Gr&egrave;ce, soleil">
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Im Beispiel werden die gleichen Stichwörter in den Sprachen Deutsch, US-Englisch, Britisch-Englisch und Französisch definiert. Dabei wird das Attribut `lang=` benutzt, um die Sprache anzugeben. Als Angabe ist ein Sprachenkürzel (Anhang A.3) erlaubt, z.B. `de` für deutsch, `en` für englisch, `fr` für französisch, `it` für italienisch oder `es` für spanisch.

**Beachten Sie:** Das Attribut `lang=` gehört zu den Universalattributen (Kapitel 13.1) in HTML.

### **4.1.3 Angabe zum Default-Zeichensatz**

Sie können mit Hilfe einer Meta-Angabe ausdrücklich bestimmen, welchen Zeichensatz Sie innerhalb der HTML-Datei verwenden. Diese Angabe ist für den Web-Server und den Web-Browser sehr hilfreich, denn sie teilt beiden Softwaretypen eindeutig mit, nach welchem Zeichensatz die Inhalte der HTML-Datei im Zweifelsfall zu interpretieren sind. Wichtig ist die Zeichensatzangabe vor allem dann, wenn Sie innerhalb der HTML-Datei z.B. deutsche Umlaute nicht maskieren. Ein Beispiel:

```
<head>
<meta http-equiv="content-type"
  content="text/html; charset=ISO-8859-1">
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Die Angabe zum Zeichensatz wird mit `http-equiv="content-type"` definiert. Daran ist schon erkennbar, dass sich die Angabe auch an den Web-Server richtet. Bei `content=` geben Sie zunächst den Mime-Type an, der für HTML-Dateien immer `text/html` lautet. Dahinter folgt, durch einen Strichpunkt getrennt, die Angabe des Zeichensatzes. Im obigen Beispiel wird der

Zeichensatz ISO-8859-1 definiert (das ist der normale Zeichensatz für westeuropäische Sprachen, unter anderem auch für Deutsch). Erlaubt sind Zeichensatzangaben, wie sie auf der Web-Adresse [www.iana.org/assignments/character-sets](http://www.iana.org/assignments/character-sets) angegeben sind. Weitere Informationen über solche Zeichensätze, auch über andere als diesen Zeichensatz, finden Sie im Abschnitt 23.1 *Zeichensätze (ISO-8859-Familie und andere)*.

#### 4.1.4 Angaben zu Default-Sprachen für Scripts und Stylesheets

Für HTML-Ergänzungssprachen wie Stylesheets und Scripts können Sie bestimmen, welche Sprachen Sie innerhalb der HTML-Datei verwenden. Vor allem bei JavaScript ist diese Angabe wichtig, wenn Sie die HTML-Variante »Strict« benutzen. Denn das `language`-Attribut im `script`-Tag, mit dem traditionellerweise die verwendete Script-Sprache angegeben wird, ist in dieser HTML-Variante nicht erlaubt. Ein Beispiel:

```
<head>
<meta http-equiv="Content-Script-Type"
  content="text/javascript">
<meta http-equiv="Content-Style-Type" content="text/css">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Die Angaben zum Bestimmen der Default-Script-Sprache und der Default-Stylesprache haben einen einheitlichen Aufbau. Die Angabe zur bevorzugten Script-Sprache leiten Sie mit `<meta http-equiv="Content-Script-Type">` ein, die Angabe zur bevorzugten Stylesprache mit `http-equiv="Content-Style-Type"`. Bei dem Attribut `content=` müssen Sie dann den Mime-Type der gewünschten Sprache angeben. Beispiele sind `text/css` für CSS Stylesheets und `text/javascript` für JavaScript.

#### 4.1.5 Datei von Originaladresse laden

Häufig abgerufene Web-Seiten werden im Web auf so genannten Proxy-Servern zwischengespeichert. Das ist dann ein so genannter Proxy-Cache. Auch Browser speichern aufgerufene Seiten, und zwar lokal auf dem Rechner des Anwenders. Dabei spricht man vom Browser-Cache. Die Cache-Speicher sparen in vielen Fällen Leitungswege und Ressourcen. Ein Nachteil ist jedoch, dass dem Anwender möglicherweise Daten angezeigt werden, die gar nicht mehr aktuell sind, weil auf der Original-Adresse mittlerweile neue Daten liegen. Sie können mit Hilfe einer Meta-Angabe erzwingen, dass die Daten nicht aus einem Cache-Speicher serviert werden, sondern vom Original-Server. Zu empfehlen ist diese Angabe, wenn Sie die Daten einer HTML-Datei häufig ändern und neu ins Web hochladen. Ein Beispiel:

```
<head>
<meta http-equiv="expires" content="0">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit `<meta http-equiv="expires" content="0">` veranlassen Sie, dass diese HTML-Datei in jedem Fall von der Original-Adresse geladen wird (*expires = fällig werden*).

Bei `content=` können Sie anstelle der 0 in diesem Zusammenhang aber auch ein bestimmtes Datum und eine bestimmte Uhrzeit angeben. Dadurch bewirken Sie, dass die Daten dieser Datei nach dem angegebenen Zeitpunkt auf jeden Fall vom Original-Server geladen werden sollen. Datum und Uhrzeit müssen Sie im internationalen Format angeben, zum Beispiel: `content="Sat, 15 Dec 2001 12:00:00 GMT"`. Notieren Sie den Zeitpunkt so wie im Beispiel mit allen Leerzeichen, Doppelpunkten zwischen Stunden, Minuten und Sekunden sowie der Angabe GMT am Ende. Als Wochentagsnamen sind erlaubt Mon (Montag), Tue (Dienstag), Wed (Mittwoch), Thu (Donnerstag), Fri (Freitag), Sat (Samstag) und Sun (Sonntag). Als Monatsnamen sind erlaubt Jan (Januar), Feb (Februar), Mar (März), Apr (April), May (Mai), Jun (Juni), Jul (Juli), Aug (August), Sep (September), Oct (Oktober), Nov (November) und Dec (Dezember).

Anstelle der 0 können Sie auch eine Zahl angeben. Diese Zahl bedeutet dann die Anzahl Sekunden, nach deren Ablauf der Web-Browser eine Datei, die er im Cache hat, auf jeden Fall wieder vom Server lädt. Mit `content="43200"` stellen Sie beispielsweise einen Wert von 12 Stunden ein.

#### 4.1.6 Weiterleitung zu anderer Adresse (Forwarding)

Diese Angabe ist zwar ziemlich eingebürgert und sehr beliebt, weshalb sie hier auch beschrieben wird. In der HTML-Spezifikation verteuelt das W3-Konsortium diese Angabe jedoch mit der Begründung, dass die Seite dann für einige Besucher überhaupt nicht mehr anzeigbar sei. Das mutet zwar etwas seltsam an, da es seit Urzeiten mit praktisch allen Browsern erfolgreich funktioniert. Es sei an dieser Stelle aber auf die Verteuflung hingewiesen. Benutzt wird diese Angabe in der Praxis oft für das automatische Weiterleiten eines Seitenbesuchers von einer alten Adresse Ihres Web-Projekts zu einer neuen Adresse.

Ein Beispiel:

```
<head>
<meta http-equiv="refresh"
  content="5; URL=http://selfhtml.teamone.de/">
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit `<meta http-equiv="refresh" content="...">` veranlassen Sie die Weiterleitung zu einer anderen Adresse. Mit `content="5;` bestimmen Sie, nach wie viel Sekunden die Weiterleitung starten soll. Die 5 im Beispiel bedeutet also, dass die aktuelle Seite, nachdem sie geladen ist, 5 Sekunden lang angezeigt wird. Danach wird die Adresse aufgerufen, die mit `url=...` angegeben wird. Notieren Sie den gesamten Befehl inklusive der etwas ungewohnten Stellung der Anführungszeichen so wie im Beispiel oben angegeben. Setzen Sie Ihre gewünschte Anzeigedauer und den Namen der aufzurufenden Adresse ein. Bei lokalen Adressen auf dem gleichen Server können Sie absolute oder relative Pfadangaben ohne Angabe von `http://` und Domain notieren. Bei Weiterleitung zu Dateien im gleichen Verzeichnis genügt der Dateiname. Bei einem Timeout von 0 wird die angegebene nächste Datei sofort geladen.

**Beachten Sie:** Verlassen Sie sich nicht darauf, dass diese Angabe funktioniert. Wenn Sie etwa eine Weiterleitung zu einer neuen Adresse eingeben, notieren Sie am besten auf der gleichen Seite zur Sicherheit noch einen normalen Verweis zu der neuen Adresse. So finden auch Anwender, bei denen die Meta-Angabe nicht funktioniert, über den Verweis den Weg zu Ihrer neuen Heimat.

### 4.1.7 Angaben zu PICS

PICS steht für »Platform for Internet Content Selection«. Es handelt sich dabei um ein standardisiertes Schema zum Bezeichnen von Internet-Inhalten. Das Schema wurde ins Leben gerufen, um jugendfreie Internet-Inhalte als solche auszuzeichnen. Es gibt so genannte PICS-Labels, also Auszeichnungen. Anbieter von Web-Seiten können ihre Seiten entweder freiwillig und nach eigenem Ermessen mit einem PICS-Label versehen, oder – und das ist wohl eher Sinn der Sache – sie nehmen einen unabhängigen PICS-Service in Anspruch, der PICS-Zertifikate für Inhalte ausstellt.

Auf Einzelheiten des PICS-Systems wird hier nicht näher eingegangen. Informationen im Web finden Sie auf der PICS-Seite des W3-Konsortiums ([www.w3.org/PICS/](http://www.w3.org/PICS/)). Moderne Browser können PICS-Inhalte erkennen. Dazu lassen sich softwareseitig Einstellungen vornehmen, zum Beispiel durch Eltern von surfenden Kindern. So kann der Browser bei Seiten mit PICS-Labels durch Vergleichen der Konfiguration mit dem Label erkennen, ob der Inhalt angezeigt werden darf oder nicht. Ein Beispiel:

```
<head>
<meta http-equiv="PICS-Label" content='
(PICS-1.1 "http://www.gcf.org/v2.5"
  labels on "1994.11.05T08:15-0500"
  until "1995.12.31T23:59-0000"
  for "http://w3.org/PICS/Overview.html"
  ratings (suds 0.5 density 0 color/hue 1))'>
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Durch die Meta-Angabe `http-equiv="PICS-Label"` können Sie eine PICS-Auszeichnung für eine HTML-Datei notieren. Hinter `content=` folgt die PICS-Auszeichnung. Die gesamte Auszeichnung wird in einfache Anführungszeichen gesetzt. Die Syntax der PICS-Auszeichnung wird innerhalb davon in runde Klammern eingeschlossen. Innerhalb der runden Klammern stehen zunächst die Pflichtangabe zur PICS-Version und die Angabe des URI, wo die Inhaltskategorisierung definiert wird, auf die hier Bezug genommen wird. Die Angaben `labels on`, `until` und `for` sind optional. Entscheidend ist aber auf jeden Fall die Angabe `ratings`, in der die eigentliche Inhaltskategorisierung notiert wird. Die Kategorisierung wird dabei nochmals in runde Klammern eingeschlossen. Die einzelnen dort möglichen Angaben und ihre Bedeutung muss der PICS-Service bereitstellen, der die PICS-Zertifikate verteilt.

### 4.1.8 Meta-Angaben in separater Datei (Profil)

Sie können ein externes Profil für Meta-Angaben angeben, an das Sie sich halten. Das Profil richtet sich sowohl an Web-Browser als auch an Suchdienste. Es kann sich dabei um allgemein bekannte Profile handeln, die bestimmte Internet-Adressen als Heimat haben. Denkbar ist eine solche Adresse beispielsweise für Dublin-Core. Ebenso ist es aber auch möglich, eine eigene Datei mit eigenen Meta-Definitionen anzugeben. Ein Beispiel:

```
<head profile="http://meine.org/profil.dat">
  <meta name="Kategorie" content="A.1.B">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Um ein Profil einzubinden, notieren Sie im einleitenden `<head>`-Tag das Attribut `profile=`. Als Wert weisen Sie einen URI zu, unter dem die Profildatei abgelegt ist. Es gibt keine Angaben darüber, wie diese Profildatei auszusehen hat. Das W3-Konsortium möchte die angegebene Adresse auch eher als »global eindeutigen Namen« verstanden wissen. Web-Browser oder Suchmaschinen, die das entsprechende Profil kennen, brauchen dann gar nichts auslesen, sondern erfahren auf diese Weise einfach, dass die nachfolgenden Meta-Angaben diesem Profil gemäß zu interpretieren sind.

Im Beispiel wird ein Profil zugrunde gelegt, das auf der Internet-Adresse `http://meine.org/profil.dat` abgelegt ist. Es wird angenommen, dass in dem Profil eine Meta-Eigenschaft mit dem Namen `Kategorie` definiert wird. Diese Eigenschaft kann dann mit `<meta name="Kategorie" ...` benutzt werden.

### 4.1.9 Schema zu einer Meta-Angabe

Dies ist, wenn Sie so wollen, eine Meta-Meta-Angabe. Für Wertzuweisungen, die Sie beim `content`-Attribut einer Meta-Angabe notieren, gibt es manchmal ein bekanntes Schema, nach dem der Wert interpretiert werden kann. Mit Hilfe eines zusätzlichen Attributs können Sie angeben, nach welchem Schema die Wertzuweisung zu interpretieren ist. Ein Beispiel:

```
<head profile="http://meine.org/profil.dat">
  <meta name="Typ" scheme="MimeType" content="image/svg+xml">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

In dem Beispiel wird ein Profil zugrunde gelegt, das auf der Internet-Adresse `http://meine.org/profil.dat` abgelegt ist. Es wird angenommen, dass in dem Profil eine Eigenschaft mit dem Namen `Typ` definiert wird. Diese Eigenschaft kann dann in einer Meta-Angabe mit `name="Typ"` bezeichnet werden. Die Angabe zu `scheme=` im gleichen Tag stellt den Versuch dar, der auslesenden Software einen Hinweis zu geben, wie die zugehörige Angabe bei `content=` zu verstehen ist. Im Beispiel wird mit `scheme="MimeType"` versucht, der Software mitzuteilen, dass die Angabe bei `content=` einen Mime-Type (Anhang A.2) meint.



### 4.1.10 Diverse Meta-Angaben

Die folgenden Meta-Angaben sind verstreute »Sammlerstücke«. Oft sind es Angaben, die mit bestimmten Web-Servern oder Browsern oder Suchmaschinen-Robots funktionieren, oder Angaben, die von Bearbeitungs-Software eingetragen wird. Manchmal sind es aber möglicherweise auch nur Erfindungen von Leuten, die an magische Kräfte in ihren HTML-Dateien glauben. Hier eine kleine Auswahl solcher Meta-Angaben:

- `<meta http-equiv="content-language" content="de">`: Sprache des Dateiinhalts (HTTP 1.0 und RFC1766)
- `<meta name="generator" content="MS Frontpage 2000">`, `<meta name="generator" content="Netscape Composer">`: Wird von HTML-Editoren, vor allem von Wysiwyg-Editoren ohne Wissen des Autors eingefügt. Softwarehersteller können so prima das Web durchsuchen und feststellen, wo und wie viel mit ihrer Software gearbeitet wird.
- `<meta http-equiv="ext-cache" content="name=/some/path/index.db; instructions=user instructions">`: Anweisung an den Netscape-Browser: nicht den normalen lokalen Browser-Cache benutzen, sondern den angegebenen externen Cache.
- `<meta http-equiv="cache-control" content="no-cache">`: Anweisung an den Browser: keinen Cache benutzen, sondern von Originalseite laden.
- `<meta http-equiv="set-cookie" content="cookievalue=xxx;expires=friday, 31-dec-99 23:59:59 gmt; path=/;">`: Cookie setzen (Netscape).
- `<meta http-equiv="pragma" content="no-cache">`: An Proxy-Agenten: Datei bitte nicht auf Proxy-Server speichern!
- `<meta name="revisit-after" content="20 days">`: An Such-Robot: bitte nach 20 Tagen erneut vorbeikommen und Datei neu auslesen!

### 4.1.11 Adressbasis und Zielfensterbasis



#### *Adressbasis*

Sie können innerhalb einer HTML-Datei, die Sie auf einem WWW-Server ablegen, nochmals deren internetweit eindeutigen, genauen URI notieren. Ein Web-Browser, der diese Information ausliest, kann in Fehlersituationen besser auf verknüpfte oder referenzierte Dateien zugreifen. Bei Projektverweisen zu anderen HTML-Dateien und bei Referenzen von Grafiken oder Multimedia gilt die hier definierbare Basis als Bezug. Ein Beispiel:

```
<head>
<base href="http://selfaktuell.teamone.de">
<!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Die Angabe der Dateibasis erfolgt im Dateikopf mit `<base href= ... >` und dem genauen URI der Datei (*base* = *Basis*, *href* = *hyper reference* = *Hyper(text)-Referenz*). Wenn nun in der Datei beispielsweise die Grafikreferenz `` steht, so ermittelt der WWW-

Browser diese Grafik mit dem absoluten URI ``.

**Beachten Sie:** Wenn es wichtig für Sie ist, dass Ihre Projekte leicht auf andere Server-Rechner, in andere Verzeichnisstrukturen usw. übertragbar bleiben, sollten Sie auf die Definition einer Adressbasis verzichten. Denn mit der Angabe einer absoluten Adressbasis wird der WWW-Browser stets versuchen, Verweisziele und referenzierte Dateien von der hier angegebenen Adresse zu laden. Dies macht das Austesten der Dateien auf einem lokalen PC unmöglich!

Die Angabe der Adressbasis ist dagegen ein gewisser Schutz vor HTML-unkundigen Anwendern, die sich eine HTML-Datei lokal abspeichern oder das Cache-Verzeichnis ihres WWW-Browsers durchsuchen. Beim lokalen Aufruf der HTML-Datei wird der WWW-Browser stets eine Online-Verbindung verlangen und die Datei von der angegebenen Adresse laden wollen.



### Zielfensterbasis

Diese Angabe ist z.B. in Verbindung mit Frames sinnvoll. Sie können für eine HTML-Datei, die innerhalb eines Framesets in einem Frame-Fenster angezeigt wird, festlegen, dass alle Verweise dieser Datei in einem bestimmten anderen Frame-Fenster angezeigt werden, solange bei einem Verweis kein anderes Frame-Fenster angegeben wird. Da häufig alle Verweisziele einer Datei, die in einem Frame-Fenster angezeigt wird, in einem bestimmten anderen Frame-Fenster angezeigt werden sollen, spart diese einmalige Angabe im Dateikopf viel Tipparbeit und hilft, den Dateiumfang zu verkleinern. Siehe auch das Kapitel 9.3 *Verweise bei Frames*. Ein Beispiel:

```
<head>
<base target="RechtesFenster">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit `<base target= ... >` legen Sie das Default-Fenster fest, in dem Verweisziele angezeigt werden sollen. Voraussetzung ist im obigen Beispiel, dass Sie ein Frameset mit Frame-Fenstern definiert haben. Bei dem gewünschten Frame-Fenster müssen Sie mit dem Attribut `name=` den Fensternamen vergeben haben, den Sie hier bei `<base target= ... >` angeben.

Anstelle eines selbst vergebenen Frame-Fensternamens können Sie aber auch eine der folgenden festen Angaben notieren, die auch für HTML-Dateien außerhalb von Framesets interessant sind:

- `<base target="_blank">` öffnet jeden Verweis der Datei in einem neuen Browser-Fenster.
- `<base target="_top">` öffnet jeden Verweis der Datei im gesamten Browser-Fenster und befreit die Anzeige aus allen eventuell angezeigten Framesets (z.B. auch aus fremden, feindlichen Framesets – siehe Abschnitt 9.5.2 über Schaufenstereffekte).
- `<base target="_parent">` öffnet jeden Verweis der Datei im übergeordneten Frameset und befreit die Anzeige dem inneren Frameset. `_parent` und `_top` sind dann verschieden, wenn ein Frame-Fenster nochmals ein komplettes Frameset enthält, also eher etwas für Mikro-Freaks.

### 4.1.12 Logische Beziehungen



#### Beziehungen zu anderen Quellen

Hinweis für HTML-Einsteiger: Das hier beschriebene Element hat nichts mit normalen, anklickbaren Verweisen innerhalb einer HTML-Datei zu tun. Solche Verweise werden im Kapitel 6 über *Verweise* beschrieben.

Die auf den folgenden Seiten beschriebenen Möglichkeiten sind zwar seit Urzeiten Bestandteil von HTML, werden aber leider von keinem der populären heutigen Web-Browser unterstützt und sind von daher zunächst einmal wertlos. Sie stellen jedoch eine hervorragende Möglichkeit dar, um intern die hypertextuellen Abhängigkeiten einer HTML-Datei, etwa die vorherige und nächste Seite, die übergeordnete Seite usw. explizit anzugeben. Suchdienste oder Softwareprogramme können dann beispielsweise ganze »Site-Maps« Ihres Projekts erstellen, also grafische Darstellungen der Seitenstruktur. Noch machen zwar kaum Suchmaschinen davon Gebrauch, doch es ist zu hoffen, dass sich dies noch ändern wird.

Web-Browser sollten, wenn sie solche Tags in einer HTML-Datei vorfinden, eine entsprechende Button-Leiste oder vergleichbare Bedienelemente anzeigen. Mit dem Anklicken der Buttons sollte der Anwender dann Sprungverweise zu den verknüpften Verweiszielen ausführen. Weder Netscape noch der MS Internet Explorer sind bis heute auf diese sinnvollen Vorschläge eingegangen – wenn also einer dieser Hersteller behauptet, sein Browser interpretiere HTML 4.0 vollständig, dann dürfen Sie schmunzelnd darauf hinweisen, dass der Browser nicht mal HTML 2.0 vollständig interpretiert – eben wegen dieser Tags. Sie können pro HTML-Datei ein Set solcher Tags notieren. Sie können alle, aber auch nur einzelne, gerade benötigte Tags verwenden. Ein Beispiel:

```
<head>
<link rel="contents" href="inhalt.htm" title="Inhaltsverzeichnis">
<link rel="index" href="stichwrt.htm" title="Stichwortverzeichnis">
<link rel="glossary" href="glossar.htm"
  title="Begriffs-Glossar">
<link rel="next" href="augsburg.htm"
  title="nächste Seite">
<link rel="previous" href="aachen.htm" title="vorherige Seite">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit `<link rel=` leiten Sie eine logische Dateibeziehung ein. Dahinter folgt der Typ der Dateibeziehung (*link* = *Verweis*, *rel* = *relationship* = *Verwandtschaft*). Erlaubte Typen sind:

- `rel="contents"` steht für: Bezug zum Inhaltsverzeichnis (*contents* = *Inhaltsverzeichnis*)
- `rel="chapter"` steht für: Bezug zum Kapitel (*chapter* = *Kapitel*)
- `rel="section"` steht für: Bezug zum Abschnitt (*section* = *Abschnitt*)
- `rel="subsection"` steht für: Bezug zum Unterabschnitt (*subsection* = *Unterabschnitt*)
- `rel="index"` steht für: Bezug zum Stichwortverzeichnis
- `rel="glossary"` steht für: Bezug zum Glossar
- `rel="appendix"` steht für: Bezug zum Anhang (*appendix* = *Anhang*)

- `rel="copyright"` steht für: Bezug zur Copyright-Angabe
- `rel="next"` steht für: Bezug zur nächsten Datei in der »Guided tour« (*next = nächste Seite*)
- `rel="prev"` steht für: Bezug zur vorherigen Datei in der »Guided tour« (*prev = previous = vorherige Seite*)
- `rel="start"` steht für: Bezug zur ersten Datei in der »Guided tour« (*start = erste Seite*)
- `rel="help"` steht für: Bezug zum Hilfekontext (*help = Hilfe*)
- `rel="bookmark"` steht für: Bezug zu einem allgemeinen Orientierungspunkt (*bookmark = Lesezeichen*)
- `rel="stylesheet"` steht für: Bezug zu einer Datei mit Style-Definitionen. Diese eine Angabe wird von Netscape und dem MS Internet Explorer tatsächlich interpretiert, und zwar, um Stylesheets in HTML einzubinden.
- `rel="alternate"` steht für: Bezug zu einer Datei mit dem gleichen Inhalt wie der aktuellen, jedoch in einer anderen Dokumentversion (*alternate = alternierend*). Von diesem Typ können Sie bei Bedarf auch mehrere Befehle notieren.

Das Attribut `href=` dient zur Bestimmung, wohin ein Verweis zum Verknüpfungsziel führen soll. Bei der Wertzuweisung gelten die gleichen Regeln wie bei normalen Verweisen. Mit dem Attribut `title=` können Sie die Beschriftung definieren, die verwendet wird, wenn der Browser Verweis-Buttons anzeigt. Die Angabe sollte das Verknüpfungsziel bezeichnen. Weitere mögliche Attribute im `<link>`-Tag sind:

- `media=` zur Bestimmung des Ausgabemediums (mehr dazu im Abschnitt 15.2.4 *Verschiedene separate Stylesheets für unterschiedliche Ausgabemedien*).
- `target=` zur Angabe des Zielfensters, in dem das Verknüpfungsziel ausgegeben werden soll. Bei Frames kann das eines der definierten Frame-Fenster sein. Es kann aber auch einer der reservierten Fensteramen angegeben werden: `target="_blank"` öffnet den Verweis in einem neuen Browser-Fenster, `target="_top"` im gesamten Browser-Fenster und befreit die Anzeige aus allen eventuell angezeigten Framesets, `target="_parent"` im übergeordneten Frameset und befreit die Anzeige aus dem inneren Frameset.
- `hreflang=` erlaubt die Angabe der Landessprache des Verknüpfungsziels mit Hilfe eines Sprachenkürzels.
- `charset=` ermöglicht die Angabe des Zeichensatzes, den das Verknüpfungsziel verwendet, z.B. ISO-8859-1 – erlaubt sind dabei Zeichensatzangaben, wie sie auf der Web-Adresse <http://www.iana.org/assignments/character-sets> angegeben sind.

Ferner kann das `link`-Element Universalattribute haben.

**Beachten Sie:** Neben diesen im HTML-Standard verankerten Ausprägungen des `link`-Elements gibt es noch einen weitere, die zum Einbinden downloadbarer Schriftarten in HTML dient. Ein Beispiel:

```
<link rel="fontdef"
src="http://www.meine.com/fonts/chianti.pfr">
```

Damit binden Sie eine Schriftart namens `chianti.pfr` ein. Näheres zu diesem Thema erfahren Sie im Abschnitt 23.4 *Downloadbare Schriftarten*.



### Rückbeziehungen

Rückbeziehungen sind für »bidirektionale« Beziehungen zweier HTML-Dateien gedacht. Das ist der Fall, wenn sowohl Datei A auf Datei B verweist als auch Datei B auf Datei A, wenn also beide logisch gesehen ein sinnvolles Verweisziel von der jeweils anderen sind.

Ebenso wie logische Beziehungen zu anderen Dateien sollten Rückbeziehungen als Buttons oder in vergleichbarer Form dargestellt werden (Netscape und der MS Internet Explorer bleiben auch hierbei stumm). Suchmaschinen können aus Rückbeziehungsangaben interessante Schlüsse über die Hypertextstruktur eines Projekts ziehen.

Beispiel aus *datei\_1.htm*:

```
<head>
<link rel="alternate" href="datei_2.htm"
      title="zweite Lesart">
</head>
```

Beispiel aus *datei\_2.htm*:

```
<head>
<link rev="alternate" href="datei_1.htm"
      title="erste Lesart">
</head>
```

Mit `<link rev=` leiten Sie eine Rückbeziehung ein. Ansonsten ist die Funktionalität identisch mit `<link rel=` für logische Beziehungen zu anderen Quellen. Es sind auch die gleichen anderen Attribute möglich. Im obigen Beispiel sehen Sie, wie zwei HTML-Dateien, deren angenommener Inhalt zwei verschiedene Betrachtungsweisen ein und derselben Sache behandelt, durch Rückbeziehungen logisch miteinander verknüpft werden.

#### 4.1.13



#### Durchsuchbarkeit mit Server-Kommunikation

Heute spielt dieses Element in der Praxis keine Rolle mehr, da moderne WWW-Browser bequemes und schnelles Suchen nach Text in einer angezeigten HTML-Datei erlauben. Das Element ist mittlerweile auch als *deprecated* eingestuft und soll künftig aus dem HTML-Standard entfallen. Die folgenden Informationen haben daher eher »historischen« Charakter.

Eine HTML-Datei kann eine durchsuchbare Datei sein. Das bedeutet, dass ihr Inhalt auf dem Server-Rechner nach allen vorkommenden Wörtern indexiert wurde. Es hat jedoch nur einen Sinn, eine HTML-Datei als durchsuchbar zu definieren, wenn ihr Inhalt tatsächlich in indizierter Form, also in einer Suchindexdatei auf dem Server-Rechner vorliegt.

Bei einer als durchsuchbar definierten HTML-Datei erscheint bei der Anzeige im Web-Browser am Anfang der Datei ein Eingabefeld, das dem Anwender die Eingabe eines Suchbegriffs erlaubt.

Ein Beispiel:

```
<head>
<isindex prompt="Geben Sie einen Suchbegriff ein: ">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Mit `isindex` deklarieren Sie eine HTML-Datei als durchsuchbar. Mit dem Attribut `prompt=` bestimmen Sie einen Eingabeaufforderungstext. Der Text muss in Anführungszeichen stehen (*prompt* = Aufforderung).

## 4.2 Dateiweite Einstellungen

### 4.2.1 Farben für Hintergrund, Text und Verweise



#### Farbe für den Hintergrund

Sie können eine Farbe für den Hintergrund des Anzeigefensters bestimmen. Die gesamte HTML-Datei wird auf dieser Hintergrundfarbe angezeigt.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#CCFFFF">
<h1>Text, Verweise, Grafikreferenzen usw.</h1>
</body>
</html>
```

Die Angabe zur Hintergrundfarbe erfolgt im einleitenden `<body>`-Tag der HTML-Datei. Mit dem Attribut `bgcolor=` bestimmen Sie die Farbe für den Bildschirmhintergrund (*bgcolor* = *background color* = Hintergrundfarbe). Die gesamte HTML-Datei wird auf der hier definierten Hintergrundfarbe angezeigt. Für die Farbauswahl gelten die Regeln zum Definieren von Farben in HTML.

**Beachten Sie:** Das Attribut `bgcolor=` ist als *deprecated* eingestuft und soll künftig vermieden werden. Den gleichen Effekt erreichen Sie nämlich auch mit Hilfe von CSS Stylesheets, z.B. so:

```
<body style="background-color:#CCFFFF">
```

Wenn Sie ein Dokument aus mehreren HTML-Dateien erstellen, das einen einheitlichen Hintergrund und Textvordergrund haben soll, müssen Sie die Hintergrundfarbe in jeder HTML-Datei neu definieren. Natürlich können Sie auch für jede HTML-Datei andere Farben definieren. Wenn Sie eine Hintergrundfarbe definieren, sollten Sie auch passende (kontrastierende) Textvordergrundfarben definieren (siehe hierzu folgender Abschnitt).



#### Farben für Text und Verweise

Sie können dateiweite Farben definieren:

- für Text (gültig für alle Elemente wie Überschriften, normalen Fließtext, Listen usw.),

- für Verweise zu noch nicht besuchten Stellen,
- für Verweise zu bereits besuchten Stellen,
- für Verweise, während sie angeklickt werden.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#663333" text="#FFCC99"
  link="#FF9966" vlink="#FF9900" alink="#FFFFFF">
<h1>Text</h1>
<a href="http://www.yahoo.de/">Verweis zu Yahoo</a>
</body>
</html>
```

Die Angaben erfolgen im einleitenden `<body>`-Tag der HTML-Datei. Für die Farbauswahl gelten die Regeln zum Definieren von Farben in HTML.

- Mit `text=` definieren Sie eine Farbe für den Text.
- Mit `link=` definieren Sie eine Farbe für Verweise zu noch nicht besuchten Dateien (*link = Verweis*).
- Mit `vlink=` definieren Sie eine Farbe für Verweise zu bereits besuchten Dateien (*vlink = visited link = besuchter Verweis*).
- Mit `alink=` definieren Sie eine Farbe für Verweise, die der Anwender gerade anklickt (*alink = activated link = aktivierter Verweis*).

**Beachten Sie:** Alle diese Attribute sind als *deprecated* eingestuft und sollen künftig vermieden werden. Den gleichen Effekt erreichen Sie nämlich auch mit Hilfe von CSS Stylesheets, z.B. im Dateikopf zwischen `<head>` und `</head>` mit dem folgenden Quelltext:

```
<style type="text/css">
body { background-color:#663333; color:#FFCC99; }
a:link { color:#FF9966; }
a:visited { color:#FF9900; }
a:active { color:#FFFFFF; }
</style>
```

Die definierten Textvordergrundfarben sollten mit der definierten Hintergrundfarbe kontrastieren. Wenn Sie z.B. eine dunkle Hintergrundfarbe definieren, sollten Sie helle Textvordergrundfarben wählen (z.B. Weiß, Gelb, Hellgrün und Hellblau).

### Weitere Möglichkeiten

Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15 nach, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden.

Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: background-color, color, Pseudoformate für Verweise. Das `<body>`-Tag kann auch Universalattribute enthalten.

## 4.2.2 Hintergrundbild (Wallpaper)

Sie können für die Anzeige einer HTML-Datei ein Hintergrundbild bestimmen. Dabei wird das Bild über das ganze Anzeigefenster hinweg immer wiederholt, sodass ein Tapeteneffekt (Wallpaper) entsteht. Besonders geeignet für Wallpaper-Effekte sind relativ kleine Grafiken, die irgendein abstraktes Muster darstellen. Typische Beispiele finden Sie im Kapitel über Grafiken im Abschnitt 24.2.4 *Wallpapers (Hintergrundbilder)*. Die Hintergrundgrafik sollte als Grafikdatei im GIF-Format oder JPEG-Format vorliegen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body background="background.jpg" text="#990000"
  link="#0000CC" vlink="#000066" alink="#000000">
<h1>Text</h1>
<a href="http://www.yahoo.de/"><b>Verweis zu Yahoo</b></a>
</body>
</html>
```

Die Angabe zum Einbinden eines Hintergrundbildes erfolgt im einleitenden `<body>`-Tag der HTML-Datei. Mit dem Attribut `background=` bestimmen Sie eine Grafikdatei als Hintergrundbild (*background = Hintergrund*). Die gesamte HTML-Datei wird auf dem hier definierten Hintergrundbild angezeigt.

**Beachten Sie:** Das Attribut `background=` ist als *deprecated* eingestuft und soll künftig vermieden werden. Den gleichen Effekt erreichen Sie nämlich auch mit Hilfe von CSS Stylesheets, z.B. so:

```
<body style="background-image:url(background.jpg)">
```



4.1: Ein Hintergrundbild sollte so beschaffen sein, dass Text trotzdem gut lesbar bleibt.



Bei der Adressierung des Hintergrundbildes gelten die Regeln zum Referenzieren in HTML. Das Hintergrundbild gilt jeweils für die HTML-Datei, in der es definiert wird. Wenn Sie ein Dokument aus mehreren HTML-Dateien erstellen, das einen einheitlichen Hintergrund haben soll, müssen Sie die Angabe in jeder HTML-Datei wiederholen. Wenn Sie ein Hintergrundbild definieren, sollten Sie auch passende Farben für Text und Verweise definieren.

Verwenden Sie bei textorientierten WWW-Seiten unauffällige Hintergrundbilder, bei denen die Lesbarkeit des Textes nicht leidet. Auffällige Hintergrundfarben sollten Sie nur verwenden, wenn die gesamte WWW-Seite grafisch aufgebaut ist und wenn die Grafiken im Vordergrund zu dem auffälligen Hintergrund passen. Netscape ab Version 4.x und MS Internet Explorer ab Version 4.x interpretieren auch animierte GIF-Grafiken als Hintergrundbilder.

Der MS Internet Explorer kennt ein zusätzliches Attribut `bgproperties="fixed"` im einleitenden `<body>`-Tag, mit dem Sie ein unbewegliches Hintergrundbild erzeugen können. Dieses Attribut gehörte jedoch nie zum HTML-Standard und lässt sich auch durch eine CSS-Angabe ersetzen, z.B. so:

```
<body style="background-image:url(background.jpg); background-attachment:fixed;">
```

### **Weitere Möglichkeiten**

Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15 nach, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: `background-image`, `background-repeat`, `background-attachment`, `background-position`, `background-color`, Pseudoformate für Verweise.

### 4.2.3 Seitenränder definieren

Bitte benutzen Sie die hier beschriebenen HTML-Attribute nur in begründeten Ausnahmefällen – sie werden hier nur der Vollständigkeit halber erwähnt. Diese Attribute gehörten nie zum HTML-Standard und sind ein typisches Beispiel dafür, welcher Wirrwarr entsteht, wenn Browser ihr eigenes Süppchen kochen. Benutzen Sie auf jeden Fall CSS Stylesheets zum Definieren von Seitenrändern. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body marginwidth="50" marginheight="100"
  topmargin="100" leftmargin="50">
<h1>Text</h1>
viel Text usw.
</body>
</html>
```

Die Angabe für Seitenränder erfolgt im einleitenden `<body>`-Tag der HTML-Datei. Mit `leftmargin=` bestimmen Sie den Abstand zwischen linkem bzw. rechtem Fensterrand und Inhalt der Datei (*leftmargin* = linker Rand) für den Internet Explorer. Mit `topmargin=` bestimmen Sie den Abstand zwischen oberem bzw. unterem Fensterrand und dem Inhalt der Datei (*topmargin* =

oberer Rand) für den Internet Explorer. Mit `marginwidth=` und `marginheight=` bestimmen Sie die entsprechenden Abstände für Netscape. Alle Angaben erfolgen in Pixeln. Linke und rechte Ränder sind immer gleich groß, ebenso obere und untere.



4.2: Die Seitenränder lassen sich bei Netscape und Internet Explorer einstellen – die Attribute sind aber nicht HTML-konform.

**Beachten Sie:** Die definierten Seitenränder gelten jeweils für die HTML-Datei, in der sie definiert werden. Wenn Sie ein Projekt aus mehreren HTML-Dateien erstellen, das einheitliche Seitenränder haben soll, müssen Sie die Seitenränder in jeder HTML-Datei neu definieren. Natürlich können Sie auch für alle Dateien unterschiedliche Seitenränder definieren.

### Seitenränder mit CSS Stylesheets

Hier noch ein Beispiel, wie Sie die gleichen Seitenränder wie im obigen Beispiel auch mit Hilfe von CSS Stylesheets erreichen können:

```
<head>
<style type="text/css">
body { margin-left:50px; margin-right:50px;
      margin-top:100px; margin-bottom:100px }
</style>
</head>
```

Mit den obigen Angaben im HTML-Dateikopf erreichen Sie die gleiche Wirkung. Diese Angaben werden von Netscape seit Version 4.0 und vom MS Internet Explorer seit Version 3.0 interpretiert. Außerdem sind diese Angaben vom W3-Konsortium standardisiert.

#### 4.2.4 Hintergrundmusik

Sie können bestimmen, dass beim Aufruf einer HTML-Datei eine Hintergrundmusik ertönt. Dazu gibt es zwei Lösungen. Beide der im folgenden beschriebenen Lösungen – die eine für Microsofts Internet Explorer, die andere für Netscape – sind proprietär. Dazu kommt, dass Netscape 4.x in einigen Zwischenversionen dabei versagt. Beide Lösungen gehören nicht zum HTML-Standard.

Ferner sollten Sie sich beim Wunsch, Hintergrundmusik einzubinden, darüber im Klaren sein, dass die Mehrzahl der Anwender im Web genervt ist von dem Gedudel und sehr schnell wieder von solchen Seiten verschwindet. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
<!-- Microsoft: -->
<bgsound src="background.mid" loop="infinite">
</head>
<body>
<!-- Netscape: -->
<embed src="background.mid" autostart="true"
  loop="true" hidden="true" height="0" width="0">
<h1>Inhalt der Seite</h1>
</body>
</html>
```

Mit `<bgsound ...>` definieren Sie im Kopf einer HTML-Datei eine Hintergrundmusik nach Microsoft-Syntax für die Anzeige der Datei (*bgsound* = *background sound* = *Hintergrundmusik*). Mit `<embed ...>` erreichen Sie das Gleiche für Netscape (*embed* = *einbetten*). Hinter der Angabe `src=` folgt in beiden Fällen die Angabe der gewünschten Musikdatei (`src` = *source* = *Quelle*). Es sollte sich möglichst um Dateien der Typen MID, AU oder WAV handeln.

Um eine sichtbare Anzeige des Abspiel-Players zu unterdrücken, sind bei der Netscape-Syntax die Angaben `hidden="true" height="0" width="0"` erforderlich (*width* = *Breite*, *height* = *Höhe*, *hidden* = *versteckt*). Auch den automatischen Start des Abspielvorgangs müssen Sie in der Netscape-Syntax explizit angeben – durch `autostart="true"`. Schließlich können Sie noch bestimmen, ob die Musikdatei nur einmal, mehrmals oder endlos oft (bis zum Aufrufen einer anderen HTML-Datei) abgespielt wird. Nach der Microsoft-Syntax können Sie mit `loop="infinite"` eine Endloswiederholung erzwingen. Wenn Sie die Wiederholungszahl begrenzen wollen, geben Sie anstelle von *infinite* einfach die gewünschte Anzahl der Wiederholungen an. Bei der Netscape-Syntax haben Sie nur die Wahl zwischen Endloswiederholung und keiner Wiederholung. Um eine Endloswiederholung zu erzwingen, notieren Sie die Angabe `loop="true"`. Ansonsten lassen Sie die Angabe einfach weg.

**Beachten Sie:** Im obigen Beispiel wird vorausgesetzt, dass sich die Musikdatei im gleichen Verzeichnis befindet wie die HTML-Datei. Wenn die Datei in einem anderen Verzeichnis steht, müssen Sie den relativen oder absoluten Pfadnamen angeben. Das funktioniert genauso wie beim Einbinden von Grafiken. Die Ausgabe einer Hintergrundmusik setzt beim Anwender natürlich entsprechende Hardware (Soundkarte, Lautsprecher) voraus. Ferner muss der Web-Browser die Möglichkeit haben, die Ausgabe der Musikdatei zu steuern.

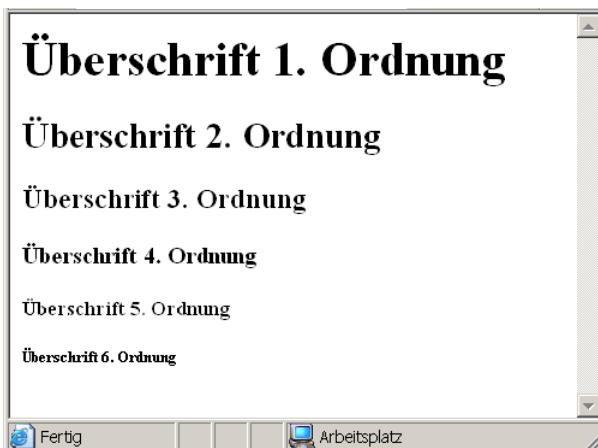
## 5 Text strukturieren

### 5.1 Überschriften

#### 5.1.1 Überschriften definieren

HTML unterscheidet sechs Überschriftenebenen, um Hierarchieverhältnisse in Dokumenten abzubilden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>&Uuml;berschrift 1. Ordnung</h1>
<h2>&Uuml;berschrift 2. Ordnung</h2>
<h3>&Uuml;berschrift 3. Ordnung</h3>
<h4>&Uuml;berschrift 4. Ordnung</h4>
<h5>&Uuml;berschrift 5. Ordnung</h5>
<h6>&Uuml;berschrift 6. Ordnung</h6>
</body>
</html>
```



5.1: HTML unterstützt sechs Hierarchieebenen bei Überschriften.

`<h[1-6]>` (*h* = *heading* = *Überschrift*) leitet eine Überschrift ein. Die Nummer steht für die Überschriftenebene. 1 ist die höchste Ebene, 6 die niedrigste. Dahinter folgt der Text der Überschrift. `</h[1-6]>` beendet die Überschrift und steht am Ende des Überschriftentextes.

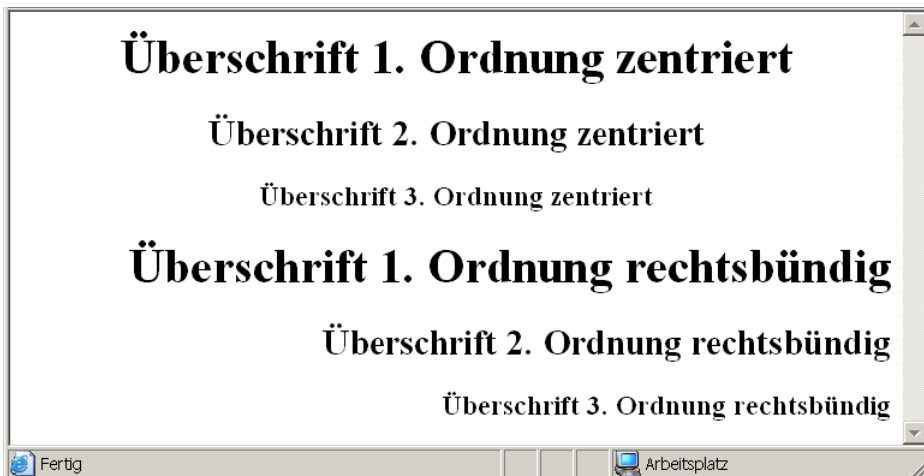
**Beachten Sie:** Die Nummern bei einleitendem und abschließendem Tag müssen gleich sein. Jede Überschrift ist ein eigener Absatz, d.h. vor und nach Überschriften sind keine Absatzschaltungen nötig. Beim Überschriftentext gelten die Aussagen zu Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen.

### 5.1.2 Überschriften ausrichten

Überschriften werden linksbündig ausgerichtet, wenn Sie nichts anderes angeben. Sie können eine Überschrift zentriert oder rechtsbündig ausrichten. Auch Blocksatz ist möglich. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1 align="center">&Uuml;berschrift 1. Ordnung zentriert</h1>
<h2 align="center">&Uuml;berschrift 2. Ordnung zentriert</h2>
<h3 align="center">&Uuml;berschrift 3. Ordnung zentriert</h3>
<h1 align="right">
    &Uuml;berschrift 1. Ordnung rechtsb&uuml;ndig
</h1>
<h2 align="right">
    &Uuml;berschrift 2. Ordnung rechtsb&uuml;ndig
</h2>
<h3 align="right">
    &Uuml;berschrift 3. Ordnung rechtsb&uuml;ndig
</h3>
</body>
</html>
```

Durch die Angabe `align="center"` im einleitenden Überschriften-Tag erreichen Sie, dass die Überschrift zentriert ausgerichtet wird (*align* = *Ausrichtung*, *center* = *zentriert*). Mit der Angabe `align="right"` wird die Überschrift rechtsbündig ausgerichtet (*right* = *rechts*). Mit `align="justify"` erzwingen Sie den Blocksatz für die Überschrift (*justify* = *justieren*). Mit der Angabe `align="left"` können Sie die Normaleinstellung (linksbündige Ausrichtung) angeben.



5.2: Überschriften dürfen Sie auch zentriert oder rechtsbündig ausrichten.

**Beachten Sie:** Nicht alle Browser beherrschen den Blocksatz. Blocksatz ist im Zusammenhang mit Überschriften auch nur bedingt praxistauglich, da Blocksatz erst bei mehrzeiligen Texten zum Tragen kommt. `align` ist im HTML-4-Standard als *deprecated* (missbilligt) eingestuft. Stattdessen wird empfohlen, CSS Stylesheets zu benutzen, z.B.:

```
<h1 style="text-align:center">...</h1>
```

Weitere Hinweise siehe folgender Abschnitt.

### 5.1.3 Überschriften formatieren mit CSS

Wie eine Überschrift vom Browser genau dargestellt wird, darauf haben Sie mit HTML keinen Einfluss. Die Browser benutzen Default-Formatierungen, um Überschriften darzustellen. Mit CSS Stylesheets können Sie Überschriften jedoch nach Wunsch formatieren. Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: Schriftformatierung, Ausrichtung und Absatzkontrolle, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder.

Im nachfolgenden Beispiel wird eine Überschrift 1. Ordnung definiert, die 36 Punkt Schriftgröße hat und in roter Farbe dargestellt wird:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1 style="font-size:36pt; color:red">
```

```
&Uuml;berschrift 1. Ordnung
</h1>
</body>
</html>
```



5.3: Mit Style Sheets formatieren Sie Überschriften nach Belieben, etwa größer und mit roter Schrift.

### Weitere Informationen

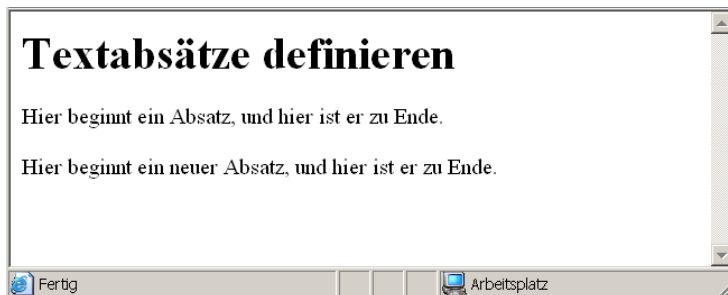
In der HTML-Referenz im Anhang finden Sie Angaben darüber, wo Überschriften vorkommen dürfen, welche anderen Elemente sie enthalten dürfen, welche Attribute sie haben können und was bei den einzelnen Attributen zu beachten ist.

## 5.2 Textabsätze

### 5.2.1 Textabsätze definieren

Absätze dienen der optischen Gliederung eines Textes. Beim Erstellen von HTML-Dateien genügt es nicht, im Editor einen harten Umbruch einzufügen. WWW-Browser ignorieren solche Umbrüche (siehe auch Kapitel 3.3, *Regeln beim Editieren von HTML*). Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Textabsätze definieren</h1>
<p>Hier beginnt ein Absatz, und hier ist er zu Ende.</p>
<p>Hier beginnt ein neuer Absatz, und hier ist er zu Ende.</p>
</body>
</html>
```



5.4: Ohne explizite Absätze zeigt HTML im Browser nur Fließtext an.

`<p>` (*p* = *paragraph* = *Absatz*) leitet einen Textabsatz ein. `</p>` beendet den Textabsatz und steht am Ende des Absatztextes.

**Beachten Sie:** Allein stehende `<p>`-Tags, wie sie früher mal zulässig waren, sind mittlerweile nicht mehr HTML-gerecht. Notieren Sie Textabsätze immer mit einleitendem und abschließendem Tag. Das `<p>`-Element darf keine anderen blockierenden Elemente wie z.B. Überschriften, Textabsätze, Listen, Zitate oder Adressen enthalten. Beim Absatztext gelten die Aussagen zu Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen.

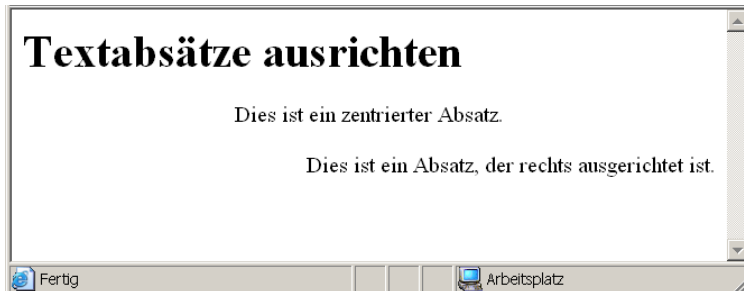
### 5.2.2 Textabsätze ausrichten

Textabsätze werden linksbündig ausgerichtet, wenn Sie nichts anderes angeben. Sie können einen Textabsatz auch zentriert oder rechtsbündig ausrichten. Auch Blocksatz ist möglich. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Textabsätze ausrichten</h1>
<p align="center">Dies ist ein zentrierter Absatz.</p>
<p align="right">
  Dies ist ein Absatz, der rechts ausgerichtet ist.
</p>
</body>
</html>
```

Durch die Angabe `align="center"` im einleitenden `<p>`-Tag erreichen Sie, dass der Textabsatz zentriert ausgerichtet wird (*align* = *Ausrichtung*, *center* = *zentriert*). Mit `align="right"` wird der Absatz rechtsbündig ausgerichtet (*right* = *rechts*). Mit `align="justify"` erzwingen Sie den Blocksatz für den Absatz (*justify* = *justieren*). Mit `align="left"` können Sie die Normaleinstellung (linksbündige Ausrichtung) angeben.





5.5: Textabsätze richten Sie mit Attributen zentriert oder rechtsbündig aus.

**Beachten Sie:** Nicht alle Browser beherrschen den Blocksatz. `align` ist im HTML-4-Standard als *deprecated* (missbilligt) eingestuft. Stattdessen wird empfohlen, CSS Stylesheets zu benutzen, z.B.:

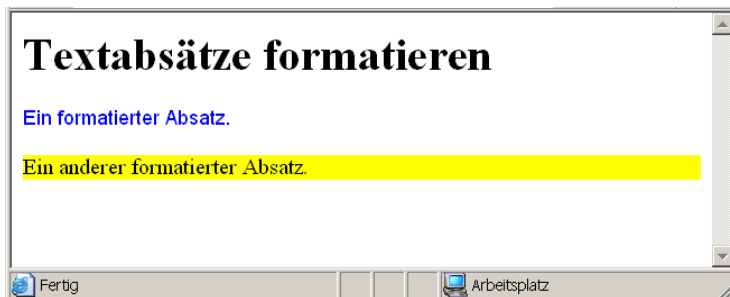
```
<p style="text-align:center">...</p>
```

Weitere Hinweise siehe folgender Abschnitt.

### 5.2.3 Textabsätze formatieren mit CSS

Wie der Text eines Absatzes genau dargestellt wird, darauf haben Sie mit HTML keinen Einfluss. Die Browser benutzen Default-Formatierungen, um den Text darzustellen. Mit CSS Stylesheets können Sie Ihre Textabsätze jedoch nach Wunsch formatieren. Bei Verwendung von CSS Stylesheets müssen Sie zunächst wissen, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: Schriftformatierung, Ausrichtung und Absatzkontrolle, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Textabsätze formatieren</h1>
<p style="font-family:Arial,sans-serif; font-size:18px; color:blue">
Ein formatierter Absatz.
</p>
<p style="background-color:yellow">
Ein anderer formatierter Absatz.
</p>
</body>
</html>
```



5.6: Mit CSS lassen sich Textabsätze auf vielfältige Weise formatieren.

Im Beispiel sind zwei Absätze definiert. Für den ersten wird bestimmt, dass er in Schriftart Arial oder einer anderen serifenlosen Schrift dargestellt wird, 18 Pixel Schriftgröße hat und in blauer Farbe erscheint. Der zweite Absatz erhält eine gelbe Hintergrundfarbe.

### Weitere Informationen

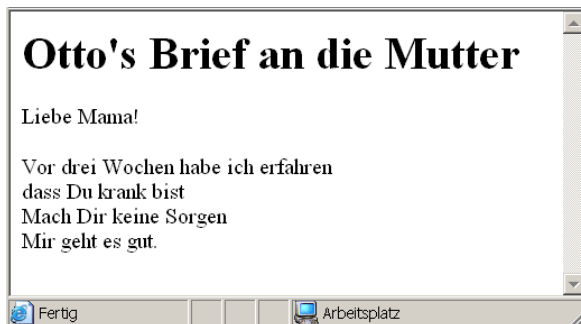
In der HTML-Referenz im Anhang finden Sie Angaben darüber, wo Textabsätze vorkommen dürfen, welche anderen Elemente sie enthalten dürfen, welche Attribute sie haben können und was bei den einzelnen Attributen zu beachten ist.

## 5.3 Zeilenumbruch

### 5.3.1 Zeilenumbruch erzwingen

Text innerhalb von normalen Absätzen, Listen sowie in Überschriften oder Tabellenzellen wird vom Web-Browser bei der Anzeige automatisch umbrochen. Sie können jedoch an einer gewünschten Stelle einen Zeilenumbruch erzwingen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Otto's Brief an die Mutter</h1>
<p>Liebe Mama!</p>
<p>Vor drei Wochen habe ich erfahren<br>
dass Du krank bist<br>
Mach Dir keine Sorgen<br>
Mir geht es gut.</p>
</body>
</html>
```



5.7: Zeilenumbrüche setzen Sie explizit mit dem Tag `<br>`.

`<br>` (*br* = *break* = *Umbruch*) fügt an der gewünschten Stelle einen Zeilenumbruch ein. Dabei ist es egal, ob das allein stehende Tag am Ende der vorherigen Zeile steht (wie im Beispiel) oder in einer eigenen Zeile, oder am Anfang der folgenden Zeile.

**Beachten Sie:** Wenn Sie XHTML-konform arbeiten, müssen Sie das `br`-Element als inhaltsleer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<br />`. Weitere Informationen dazu im Kapitel 14.1.11 zu XHTML und HTML.

### 5.3.2 Automatischen Zeilenumbruch verhindern

Sie können einen Textbereich bestimmen, in dem kein automatischer Zeilenumbruch erfolgt. Alles, was innerhalb dieses Bereichs steht, wird in einer langen Zeile angezeigt. Der Anwender kann dann mit der horizontalen Scroll-Leiste die überlangen Textzeile anzeigen. Diese Möglichkeit gehört jedoch nicht zum offiziellen HTML-Sprachstandard. Sie sollten sie daher vermeiden. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>ARD und ZDF</h1>
<nobr>Die vom ZDF sagen die vom ARD senden
st&auml;ndig Wiederholungen, und die vom ARD
sagen die vom ZDF senden st&auml;ndig Wiederholungen,
und so wiederholen sich ARD und ZDF st&auml;ndig
ohne überhaupt etwas zu senden.</nobr>
</body>
</html>
```

**Erläuterung:** `<nobr>` bewirkt, dass der auf das Tag folgende Text nicht umbrochen wird (*nobr* = *no break* = *kein Umbruch*). Am Ende des Textabschnitts, der nicht umbrochen werden soll, notieren Sie das abschließende Tag `</nobr>`.

**Beachten Sie:** Wenn Sie Textzeilen unabhängig vom Anzeigefenster des Anwenders genau kontrollieren und nach HTML-Standard arbeiten wollen, können Sie präformatierten Text (Kapitel 5.6) einsetzen.

### 5.3.3 Geschützte Leerzeichen

Sie können verhindern, dass bei einem Leerzeichen ein automatischer Zeilenumbruch erfolgen darf. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>HTML 2.0 und HTML 4.0</h1>
<p>Es gibt vieles, worin sich HTML&nbsp;2.0
und HTML&nbsp;4.0 unterscheiden.</p>
</body>
</html>
```

Die Zeichenfolge `&nbsp;` erzeugt ein geschütztes Leerzeichen (*nbsp* = *nonbreaking space* = *nicht umbrechbares Leerzeichen*). Es wird ein normales Leerzeichen angezeigt, doch an dieser Stelle kann kein Zeilenumbruch erfolgen. Notieren Sie die Zeichenfolge inklusive kaufmännisches Und am Beginn und Strichpunkt am Ende.

**Beachten Sie:** Die gleiche Wirkung erzielen Sie durch Notieren der Zeichenfolge `&#160;`. Zu dieser Art von Zeichennotation siehe auch benannte Zeichen für den Zeichensatz ISO 8859-1 im Anhang D.3. Durch Notieren mehrerer solcher benannter Zeichen hintereinander können Sie auch mehrere Leerzeichen in Folge erzwingen.

### 5.3.4 Zeilenumbruch erlauben

Web-Browser umbrechen Text normalerweise nur bei Leerzeichen, weil durch Leerzeichen Wörter voneinander abgegrenzt werden. Sie können explizit weitere Stellen markieren, an denen der Browser den Text umbrechen darf. Dies gilt für alle Absatzarten in HTML. Diese Möglichkeit gehört jedoch nicht zum offiziellen HTML-Sprachstandard. Sie sollten sie daher vermeiden. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Langes Wort</h1>
<p>Donaudampfschiffahrts-<wbr>Kapitänsmütze Donaudampfschiffahrts-
<wbr>Kapitänsmütze ... </p>
</body>
</html>
```

Mit `<wbr>` markieren Sie eine Stelle, an der getrennt werden darf, falls diese Stelle bei der Bildschirmanzeige am Ende der Zeile steht (*wbr* = *word break* = *Umbruch innerhalb eines Wortes*). Sinnvoll ist dies bei langen Wörtern oder aus Bindestrichen bestehenden Ausdrücken. Innerhalb

von Abschnitten mit verhiertem Zeilenumbruch bewirkt `<wbr>`, dass an der betreffenden Stelle trotzdem ein Umbruch erfolgen darf.

### Weitere Informationen

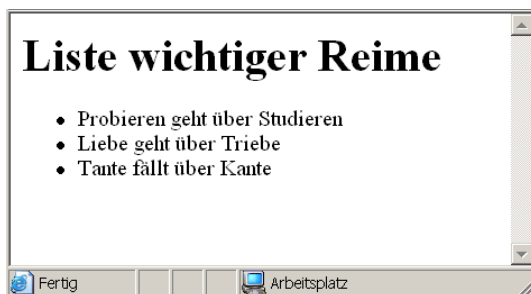
In der HTML-Referenz im Anhang finden Sie genaue Angaben darüber, wo Zeilenumbrüche vorkommen dürfen, welche Attribute sie haben können und was bei den einzelnen Attributen zu beachten ist.

## 5.4 Listen

### 5.4.1 Aufzählungsliste definieren

Aufzählungslisten sind z.B. von Bedeutung, um Produkteigenschaften oder Argumente für eine These übersichtlich darzustellen. Bei einer Aufzählungsliste werden alle Listeneinträge mit einem Aufzählungszeichen (Bullet) versehen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Liste wichtiger Reime</h1>
<ul>
<li>Probieren geht &uuml;ber Studieren</li>
<li>Liebe geht über Triebe</li>
<li>Tante f&auml;llt &uuml;ber Kante</li>
</ul>
</body>
</html>
```



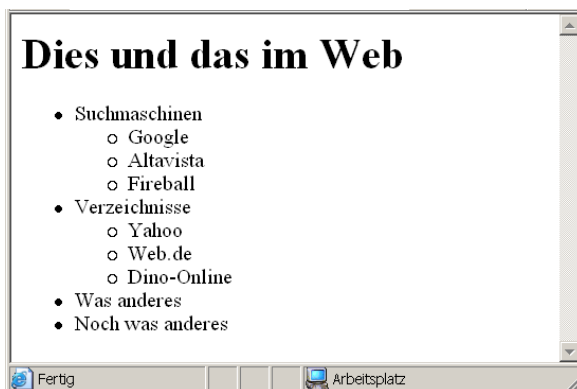
5.8: Aufzählungslisten zeigen vor den Einzelpunkten nur ein Markierungszeichen wie einen Punkt an.

`<ul>` leitet eine Aufzählungsliste ein (*ul = unordered list = unsortierte Liste*). Mit `<li>` beginnt ein neuer Punkt innerhalb der Liste (*li = list item = Listeneintrag*). `</li>` beendet den Listeneintrag. Es ist zwar nach HTML-Standard erlaubt, das abschließende `</li>`-Tag wegzulassen, doch davon ist abzuraten. `</ul>` beendet die Liste. Wie das Bullet dargestellt wird, bestimmt der Web-Browser. Das Verschachteln von Listen ist ebenfalls möglich. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Dies und das im Web</h1>

<ul>
<li>Suchmaschinen
  <ul>
    <li>Google</li>
    <li>Altavista</li>
    <li>Fireball</li>
  </ul>
</li>
<li>Verzeichnisse
  <ul>
    <li>Yahoo</li>
    <li>Web.de</li>
    <li>Dino-Online</li>
  </ul>
</li>
<li>Was anderes</li>
<li>Noch was anderes</li>
</ul>

</body>
</html>
```

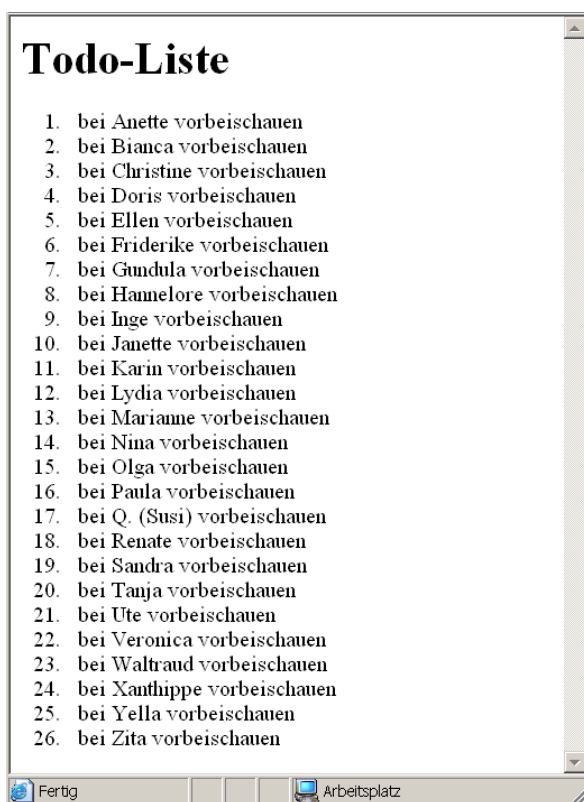


5.9: Bei verschachtelten Listen wechselt der Browser automatisch das Markierungszeichen.

Zwischen `<li>` und `</li>` darf eine komplette weitere Liste stehen. Auch andere Listentypen sind dabei erlaubt.

#### 5.4.2 Nummerierte Liste definieren

Nummerierte Listen sind z.B. von Bedeutung, um nacheinander auszuführende Aktionen oder Rangfolgen übersichtlich darzustellen. Bei einer nummerierten Liste werden alle Listeneinträge automatisch durchnummeriert.



5.10: Bei nummerierten Listen setzt der Browser automatisch Zahlen ein.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Todo-Liste</h1>
```

```

<ol>
<li>bei Anette vorbeischauen</li>
<li>bei Bianca vorbeischauen</li>
<li>bei Christine vorbeischauen</li>
<!-- usw. -->
</ol>

</body>
</html>

```

Erläuterung: `<ol>` leitet eine nummerierte Liste ein (*ol = ordered lis = nummerierte Liste*). Mit `<li>` beginnt ein neuer Punkt innerhalb der Liste (*li = list item = Listeneintrag*). `</li>` beendet den Listeneintrag. Es ist zwar nach HTML-Standard erlaubt, das abschließende `</li>`-Tag wegzulassen, doch davon ist abzuraten. `</ol>` beendet die Liste.

**Beachten Sie:** Verschachteln von nummerierten Listen ist ebenfalls möglich, bewirkt aber keine Gesamtnumerierung. Automatische Nummerierungshierarchien wie 1, 1.1, 1.1.1, sind in HTML nicht automatisch möglich.

### 5.4.3 Definitionenliste definieren

Definitionslisten sind für Glossare gedacht. Glossare bestehen aus einer Liste von Einträgen. Die Einträge eines Glossars bestehen aus einem zu definierenden Ausdruck (z.B. ein Fachbegriff) und der zugehörigen Definition.

Ein Beispiel:

```

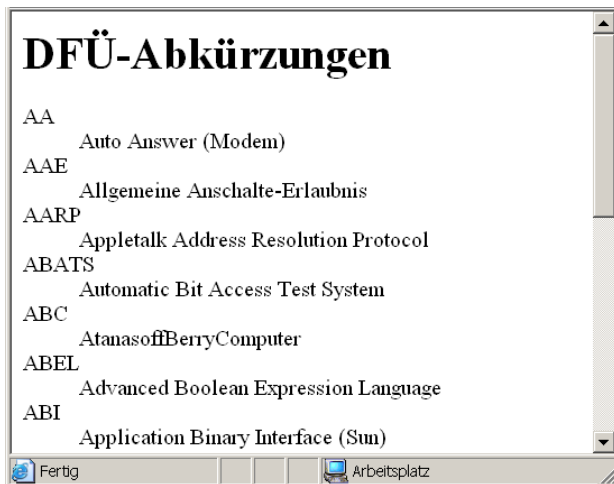
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>DF&Uuml; - Abkürzungen</h1>

<dl>
<dt>AA</dt>
<dd>Auto Answer (Modem)</dd>
<dt>AAE</dt>
<dd>Allgemeine Anschalte-Erlaubnis</dd>
<dt>AARP</dt>
<dd>Appletalk Address Resolution Protocol</dd>
<!-- usw. -->
</dl>

</body>
</html>

```





5.11: Definitionslisten bestehen aus einem Begriff und der dazugehörigen Erklärung.

`<dl>` leitet eine Definitionsliste ein (*dl* = *definition list* = *Definitionsliste*). `<dt>` leitet einen zu definierenden Ausdruck ein (*dt* = *definition (list) term* = *Ausdruck in der Definitionsliste*). `<dd>` leitet eine Definition eines Ausdrucks ein (*dd* = *definition (list) definition* = *Definition in der Definitionsliste*). `</dl>` beendet die Liste. Die End-Tags `</dt>` und `</dd>` dürfen zwar auch fehlen, doch davon ist abzuraten.

**Beachten Sie:** Das Verschachteln von Definitionslisten ist ebenfalls möglich. Dadurch können Sie baumartige Strukturen im Text abbilden. Netscape verwendet beispielsweise beim Aufbau seiner Lesezeichendatei verschachtelte Definitionslisten, um die baumartig strukturierbaren Lesezeichen zu verwalten. »Innere« Listen müssen dabei innerhalb des `dd`-Elements notiert werden. Die Abfolge von `dt`- und `dd`-Elementen innerhalb einer `dl`-Liste ist nicht streng geregelt. Es dürfen auch mehrere `dt`- oder `dd`-Elemente hintereinander folgen.

#### 5.4.4 Verzeichnis- oder Menüliste definieren

Verzeichnis- und Menülisten werden in der Praxis kaum verwendet und sind auch mittlerweile als *deprecated* gekennzeichnet, sollen also künftig aus dem HTML-Standard entfallen. Sie werden von den heutigen Browsern entweder nicht mehr oder nicht anders dargestellt als gewöhnliche Aufzählungslisten. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Verzeichnisse und Menüs</h1>
```

```

<dir>
<li>/usr/home/</li>
<li>/usr/home/web/</li>
<li>/usr/home/web/data/</li>
</dir>

<menu>
<li>Datei</li>
<li>Bearbeiten</li>
<li>Ansicht</li>
</menu>

</body>
</html>

```

Eine Verzeichnisliste leiten Sie mit `<dir>` ein und beenden Sie durch `</dir>` (*dir* = *directory* = *Verzeichnis*). `<menu>` leitet eine Menü-Liste ein, `</menu>` beendet die Menü-Liste. Mit `<li>` beginnt in beiden Listentypen ein neuer Punkt innerhalb der Liste (*li* = *list item* = *Listeneintrag*). Das End-Tag `</li>` am Ende eines Listeneintrags ist nicht zwingend erforderlich, aber zu empfehlen.

**Beachten Sie:** Verzeichnis- und Menülisen dürfen – im Gegensatz zu Aufzählungslisten – keine absatzerzeugenden Elemente zwischen `<li>` und `</li>` enthalten. Sie sind nur für zeilenartige Listen gedacht.

### 5.4.5 HTML-Eigenschaften für Listen

Es gibt eine Reihe von Attributen, die das Erscheinungsbild von Listen beeinflussen. So lassen sich beispielsweise nummerierte Listen auch alphabetisch und römisch nummerieren, und für Aufzählungslisten lässt sich der Typ des Bullet-Zeichens festlegen. Doch all diese HTML-Attribute sind mittlerweile als *deprecated* gekennzeichnet und sollen künftig aus dem HTML-Standard entfallen. Stattdessen sollten Sie Listen mit CSS formatieren (siehe unten). Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>HTML-Eigenschaften von Listen</title>
</head>
<body>
<h1>Listen mit HTML-Attributen</h1>
<h2>Aufzählungsliste</h2>
<ul type="square">
<li>Probieren geht &uuml;ber Studieren</li>
<li>Liebe geht &uuml;ber Triebe</li>
<li>Tante f&uuml;llt &uuml;ber Kante</li>
</ul>

```

```

<h2>Nummerierte Liste</h2>

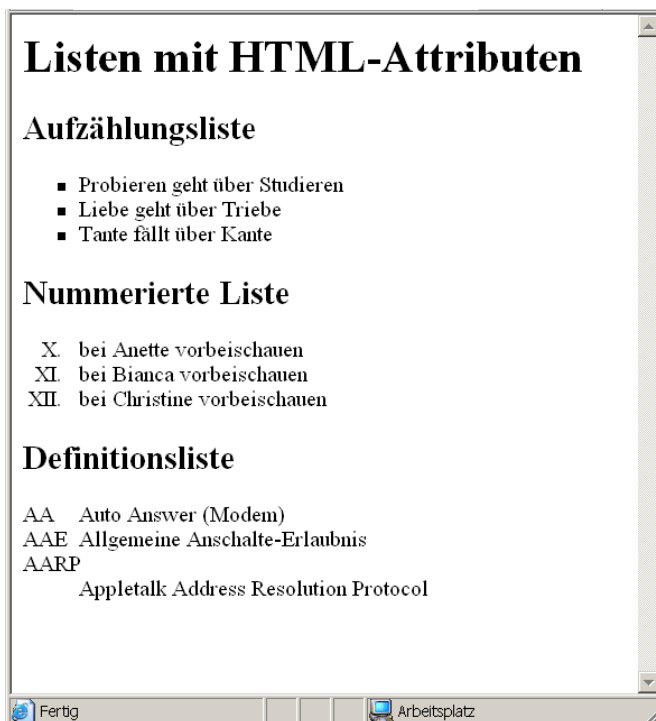
<ol type="I">
<li value="10">bei Anette vorbeischaun</li>
<li>bei Bianca vorbeischaun</li>
<li>bei Christine vorbeischaun</li>
</ol>

<h2>Definitionsliste</h2>

<dl compact>
<dt>AA</dt>
<dd>Auto Answer (Modem)</dd>
<dt>AAE</dt>
<dd>Allgemeine Anschalte-Erlaubnis</dd>
<dt>AARP</dt>
<dd>Appletalk Address Resolution Protocol</dd>
</dl>

</body>
</html>

```



5.12: Mit zusätzlichen Attributen modifizieren Sie die Listendarstellung.

Für Aufzählungslisten können Sie den Bullet-Typ bestimmen:

- Mit `<ul type="circle">` bestimmen Sie ein rundes Bullet (*circle* = *Kreis*).
- Mit `<ul type="square">` bestimmen Sie ein eckiges Bullet (*square* = *Rechteck*).
- Mit `<ul type="disc">` bestimmen Sie ein Dateisymbol als Bullet (*disc* = *Datenträger wie Disketten oder Festplatten*).

Für nummerierte Listen können Sie die Art der Nummerierung bestimmen:

- Mit `<ol type="I">` werden die Listeneinträge mit I., II., III., IV. usw. nummeriert.
- Mit `<ol type="i">` werden die Listeneinträge mit i., ii., iii., iv. usw. nummeriert.
- Mit `<ol type="A">` werden die Listeneinträge mit A., B., C. usw. nummeriert.
- Mit `<ol type="a">` werden die Listeneinträge mit a., b., c. usw. nummeriert.

Mit `start=` innerhalb des einleitenden Tags von `<ol>` können Sie einen beliebigen Startwert setzen, z.B. `<ol start="7">`. Die Nummerierung beginnt dann bei 7.

Mit `value=` innerhalb des Tags von `<li>` in einer nummerierten Liste können Sie den Startwert beliebig neu setzen, z.B. auf `<li value="15">`. Die Nummerierung fährt dann bei 15 fort.

Mit dem Attribut `compact` innerhalb des einleitenden Tags einer beliebigen Liste können Sie die kompaktere Darstellung bestimmen. Dies wurde allerdings von den Browsern noch nie oder nur vorübergehend mal interpretiert.

#### 5.4.6 Listen formatieren mit CSS

Wie eine Liste genau dargestellt wird, darauf haben Sie mit HTML keinen Einfluss. Die Browser benutzen Default-Formatierungen, um Listen darzustellen. Mit CSS Stylesheets können Sie ganze Listen und ihre Listenspunkte jedoch nach Wunsch formatieren. Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: Listenformatierung, Schriftformatierung, Ausrichtung und Absatzkontrolle, Außenrand und Abstand. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Listen mit CSS</h1>

<h2>Aufzählungsliste</h2>

<ul style="list-style-type:disk; color:red; font-weight:bold;">
<li>Probieren geht &uuml;ber Studieren</li>
<li>Liebe geht &uuml;ber Triebe</li>
```

```

<li>Tante f&auml;l;lt &uuml;ber Kante</li>
</ul>

<h2>Nummerierte Liste</h2>

<ol style="list-style-type:upper-roman; font-weight:bold;">
<li style="color:blue">bei Anette vorbeischaue</li>
<li style="color:maroon">bei Bianca vorbeischaue</li>
<li style="color:olive">bei Christine vorbeischaue</li>
</ol>

<h2>Definitionsliste</h2>

<dl style="font-size:0.7em; font-family:Verdana,Arial,sans-serif">
<dt>AA</dt>
<dd>Auto Answer (Modem)</dd>
<dt>AAE</dt>
<dd>Allgemeine Anschalte-Erlaubnis</dd>
<dt>AARP</dt>
<dd>Appletalk Address Resolution Protocol</dd>
</dl>

</body>
</html>

```

Im Beispiel wird für eine Aufzählungsliste (`<ul>...</ul>`) bestimmt, dass sie ein Disk-Symbol als Bullet-Zeichen erhält, fett und in roter Farbe erscheint. Für eine nummerierte Liste (`<ol>...</ol>`) wird bestimmt, dass sie in römischen großen Ziffern zählt und dass ihre Einträge fett dargestellt werden. Ihre einzelnen Listeneinträge werden zudem in unterschiedlichen Farben dargestellt. Für eine Definitionsliste (`<dl>...</dl>`) wird eine kleinere Schriftgröße als normal und eine bestimmte Schriftart bestimmt.

**Beachten Sie:** Netscape 4.x hat noch einige Probleme mit CSS und zeigt das obige Beispiel nicht ganz so an wie gewünscht.

### Weitere Informationen

In der HTML-Referenz im Anhang finden Sie Angaben darüber, wo die verschiedenen Listentypen vorkommen dürfen, welche anderen Elemente sie enthalten dürfen, welche Attribute sie haben können und was bei den einzelnen Attributen zu beachten ist.

## 5.5 Zitate und Adressen

### 5.5.1 Zitate definieren

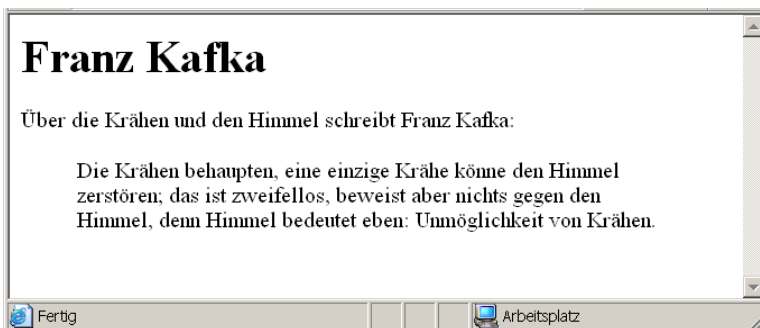
Sie können Zitate von Fremdautoren in einem eigenen, anders formatierten (zumeist eingerückten) Absatz hervorheben. Es handelt sich dabei jedoch um eine logische, inhaltliche Auszeichnung. Wie diese Absätze genau formatiert werden, bestimmt letztlich der Web-Browser. Die Vorgabe ist jedoch, solche Absätze auffällig und vom übrigen Text unterscheidbar anzuzeigen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Franz Kafka</h1>

<p>Über die Krähen und den Himmel schreibt Franz Kafka:</p>

<blockquote>
Die Krähen behaupten, eine einzige Krähe
könne den Himmel zerstören; das ist zweifellos, beweist aber nichts gegen
den Himmel, denn Himmel bedeutet eben: Unmöglichkeit von Krähen.
</blockquote>

</body>
</html>
```



5.13: Typischerweise stellen Browser ein Zitat mit eingerückten Rändern dar.

`<blockquote>` leitet einen eigenen Absatz für Zitate ein (*blockquote* = *geblocktes Zitat*).  
`</blockquote>` beendet den Absatz.

**Beachten Sie:** Das `blockquote`-Element wird gerne verwendet, um Einrückungen zu realisieren. Da es sich jedoch um eine logische Textauszeichnung handelt, die keine bestimmte Art der Formatierung vorschreibt, ist der Einrück-Effekt nicht garantiert.

### 5.5.2 Zitate mit URI-Quellenangabe

Wenn ein Zitat im Internet verfügbar ist, können Sie den URI, also die Internet-Adresse der Quelle angeben. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Die Energie des Verstehens</h1>

<p>Geben Sie nicht auf, denn Sie haben sie:</p>
<blockquote cite="http://selfhtml.teamone.de/">
Die Energie des Verstehens
</blockquote>

</body>
</html>
```

Mit dem Attribut `cite=` im einleitenden `<blockquote>`-Tag können Sie den URI der zitierten Quelle angeben (*cite* = Zitat).

**Beachten Sie:** Netscape 4.x und der MS Internet Explorer 4.x visualisieren diese Angabe nicht. Netscape 4.x kennt stattdessen eine Angabe namens `type="cite"`, die einen farbigen Querbalken am linken Rand bewirkt. Diese Angabe gehört allerdings nicht zum HTML-Standard.

### 5.5.3 Adressen definieren

Sie können Internet-Adressen von Personen oder Dateien in einem eigenen, anders formatierten (zumeist kursiv dargestellten, eingerückten) Absatz hervorheben. Auch dies ist eine logische Textauszeichnung, für deren tatsächliche Formatierung bei der Ausgabe es keine festen Vorschriften gibt. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Wenn Sie es genau wissen wollen</h1>
```

```

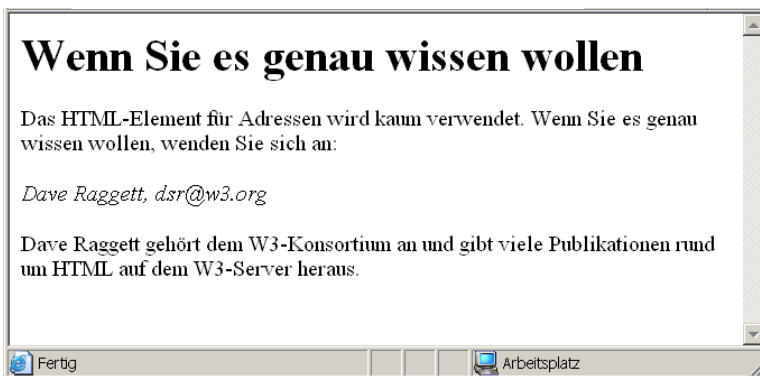
<p>Das HTML-Element für Adressen wird kaum verwendet.
Wenn Sie es genau wissen wollen, wenden Sie sich an:</p>

<address>Dave Raggett, dsr@w3.org</address>

<p>Dave Raggett gehört dem W3-Konsortium an
und gibt viele Publikationen rund um HTML auf dem W3-Server heraus.</p>

</body>
</html>

```



5.14: Adressen formatieren Browser üblicherweise als kursiven Absatz.

`<address>` leitet einen eigenen Absatz für Internet-Adressen ein. `</address>` beendet den Absatz.

**Beachten Sie:** Das `<address>`-Element darf keine anderen blockerzeugenden Elemente wie z.B. Überschriften, Textabsätze, Listen, Zitate oder Adressen enthalten. Das `<address>`-Element ist auch in Verbindung mit E-Mail-Verweisen sinnvoll.

#### 5.5.4 Zitate und Adressen formatieren mit CSS

Wie Zitate und Adressen genau dargestellt werden, darauf haben Sie mit HTML keinen Einfluss. Die Browser benutzen Default-Formatierungen. Mit CSS Stylesheets können Sie solche Elemente jedoch nach Wunsch formatieren. Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Schriftformatierung, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder.

Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>

```



```

</head>
<body>
<h1>Zitat und Adresse</h1>

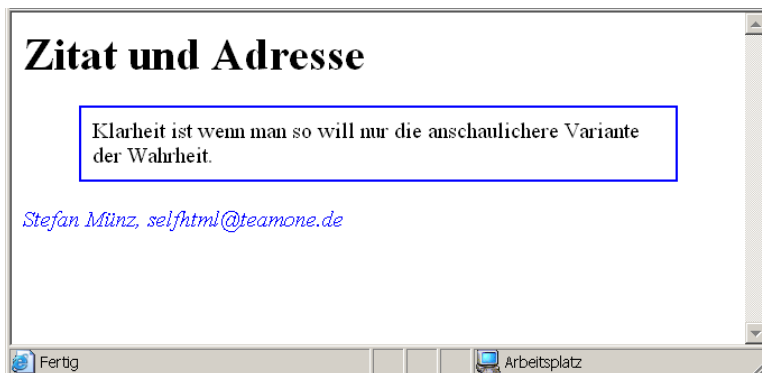
<blockquote style="padding:10px; border:thin solid blue;">
Klarheit ist wenn man so will nur die
anschaulichere Variante der Wahrheit.
</blockquote>

<address style="color:blue;">
Stefan Münz, selfhtml@teamone.de
</address>

</body>
</html>

```

Das `blockquote`-Element im Beispiel erhält einen dünnen durchgezogenen blauen Rahmen und der Text darin einen Innenabstand von 10 Pixeln zum Rahmen. Das `address`-Element erhält eine blaue Schriftfarbe.



5.15: Mit CSS lassen sich Zitate und Adressen vielfältig formatieren.

### Weitere Informationen

In der HTML-Referenz im Anhang finden Sie Angaben darüber, wo Zitate und Adressen vorkommen dürfen, welche anderen Elemente sie enthalten dürfen, welche Attribute sie haben können und was bei den einzelnen Attributen zu beachten ist.

## 5.6 Präformatierter Text (wie im Editor eingegeben)

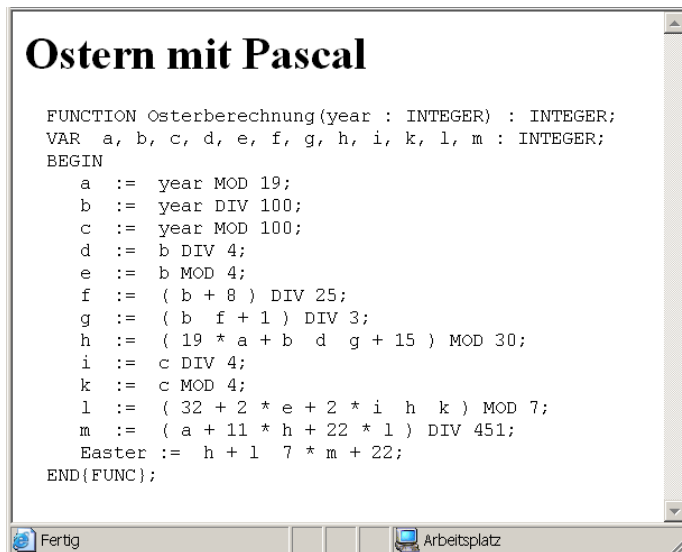
### 5.6.1 Textabschnitt mit präformatiertem Text definieren

Beispielsweise für Programmlistings ist es wichtig, dass sie in dicktengleicher Schrift dargestellt werden und dass Einrückungen so wiedergegeben werden, wie sie beim Editieren eingegeben wurden. Zu diesem Zweck bietet HTML die Möglichkeit der »präformatierten Textabschnitte« an. Auch wenn Sie tabellarische Daten darstellen müssen und auf Tabellen verzichten wollen, können Sie präformatierten Text benutzen. Und dann ist präformatierter Text auch noch dazu geeignet, um andere Elemente, beispielsweise Grafiken, auszurichten. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Ostern mit Pascal</h1>

<pre>
FUNCTION Osterberechnung(year : INTEGER) : INTEGER;
VAR  a, b, c, d, e, f, g, h, i, k, l, m : INTEGER;
BEGIN
    a := year MOD 19;
    b := year DIV 100;
    c := year MOD 100;
    d := b DIV 4;
    e := b MOD 4;
    f := ( b + 8 ) DIV 25;
    g := ( b * f + 1 ) DIV 3;
    h := ( 19 * a + b * d * g + 15 ) MOD 30;
    i := c DIV 4;
    k := c MOD 4;
    l := ( 32 + 2 * e + 2 * i * h * k ) MOD 7;
    m := ( a + 11 * h + 22 * l ) DIV 451;
    Easter := h + l * 7 * m + 22;
END{FUNC};
</pre>

</body>
</html>
```



5.16: Präformatierter Text erscheint im Browser mit allen Umbrüchen.

`<pre>` leitet einen Textabschnitt mit präformatiertem Text ein (*pre = preformatted = vorformatiert*). `</pre>` beendet den Abschnitt. Alles, was dazwischen steht, wird so angezeigt, wie es eingegeben wurde, und zwar in dicktengleicher Schrift.

**Beachten Sie:** Innerhalb von `<pre>` und `</pre>` gelten wie in anderen Textabschnitten die Aussagen zu Zeichenvorrat, Sonderzeichen und HTML-eigene Zeichen. HTML-Elemente innerhalb von `<pre>` und `</pre>` werden interpretiert. Das hat den Vorteil, dass Sie innerhalb von präformatiertem Text beispielsweise an beliebigen Stellen Grafiken einbinden können.

## 5.6.2 Breite eines Bereichs mit präformatiertem Text

Sie können eine Breite für einen Abschnitt mit präformatiertem Text erzwingen. »Breite« bedeutet dabei: die maximale Anzahl von Zeichen in einer Zeile. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>MIDI-Dateien</h1>
<pre width="80">
YBREEZE MID      17.845  15.06.97   6:01 ybreeze.mid
YGLADTR MID      34.552  15.06.97   6:01 ygladtr.mid
YHASTE MID       27.834  15.06.97   6:01 yhaste.mid
YRAG MID         18.540  15.06.97   6:01 yrag.mid

```

```
YRITMO      MID      29.056  15.06.97   6:01  yritmo.mid
           5 Datei(en)                127.827 Bytes
</pre>
</body>
</html>
```

Mit dem Attribut `width=` im einleitenden `<pre>`-Tag geben Sie eine maximale Breite für die Zeilen des Bereichs vor. Im obigen Beispiel kann keine Zeile mehr als 80 Zeichen haben. Bei längeren Zeilen kann der Browser entweder eine kleinere Schrift wählen, um die Breite einzuhalten, oder er umbricht überlange Zeilen.

**Beachten Sie:** Die Angabe zur Breite eines präformatierten Bereichs wird weder von Netscape 4.x noch vom MS Internet Explorer 4.x interpretiert und ist mittlerweile auch als *deprecated* gekennzeichnet, soll also künftig aus dem HTML-Standard entfallen.

### 5.6.3 Ältere HTML-Elemente für präformatierten Text

Der Vollständigkeit halber sollten einige ältere Elemente erwähnt werden, die allesamt nicht mehr zum HTML-Standard gehören.

- `<xmp>...</xmp>` (*xmp* = *example* = *Beispiel*) war einmal als logische Auszeichnung für Quellcode-Beispiele gedacht.
- `<plaintext>...</plaintext>` (*plaintext* = *nackter Text*) war zum Darstellen von reinem Text gedacht, etwa zum Zitieren aus einer unformatierten Textdatei.
- `<listing>...</listing>` (*listing* = *zeilenorientierter Quelltext*) war für Quelltext von Programmen gedacht.

**Beachten Sie:** Alle drei Elemente finden in der Praxis praktisch keine Verwendung mehr. Wenn Sie keine besonderen Gründe haben, sollten Sie `<pre> ... </pre>` zur Darstellung von präformatiertem Text verwenden. Der Unterschied zwischen `<xmp>...</xmp>` und `<listing>...</listing>` liegt in der Schriftweite. `<xmp>...</xmp>` bewirkt eine weitere Schrift (ausgehend von 80 Zeichen/Zeile beim Ausdruck), `<listing>...</listing>` eine engere Schrift (ausgehend von 132 Zeichen/Zeile beim Ausdruck). `<listing>...</listing>` wird von Netscape etwa auch anders dargestellt als `<pre>...</pre>`. Netscape akzeptiert zu `<plaintext>` kein abschließendes Tag `</plaintext>`, während andere Browser das End-Tag interpretieren. Netscape stellt alles in dicktengleicher Schrift dar, was Sie bis zum nächsten Block-Element (z.B. Überschrift, Textabsatz, Liste usw.) an Text eingeben.

### 5.6.4 Präformatierten Text formatieren mit CSS

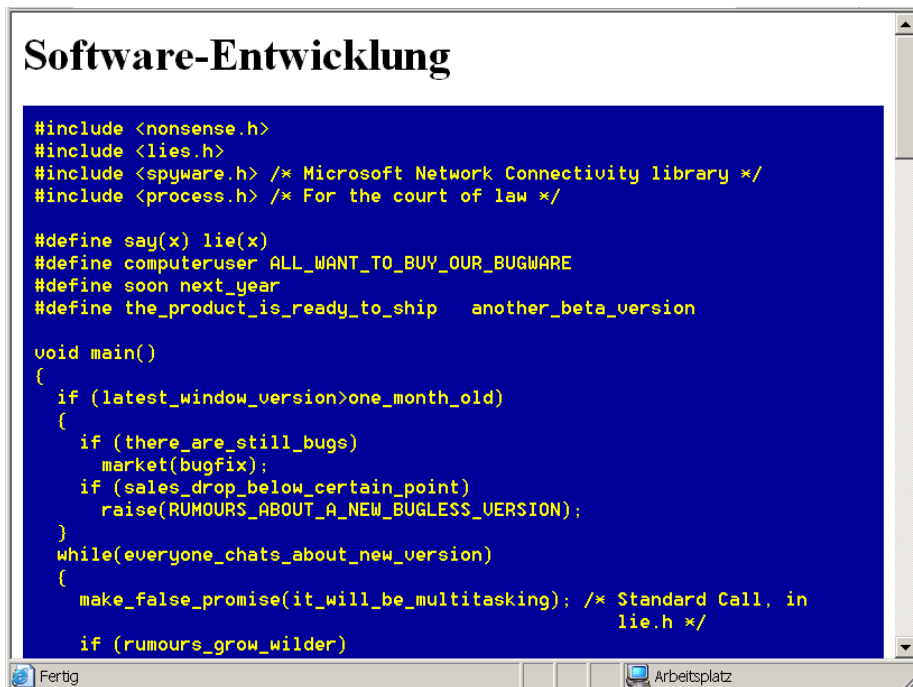
In welcher Schriftart und Schriftgröße präformatierter Text genau dargestellt werden, darauf haben Sie mit HTML keinen Einfluss. Die Browser benutzen Default-Formatierungen. Mit CSS Stylesheets können Sie ein solches Element jedoch nach Wunsch formatieren. Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15 nach, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Schriftformatierung, Aus-

richtung und Absatzkontrolle, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Software-Entwicklung</h1>

<pre style="background-color:#000099; color:#FFFF00; font-family:Fixedsys,Courier;
padding:10px;">
#include <nonsense.h>
#include <lies.h>
#include <spyware.h> /* Microsoft Network Connectivity library */
#include <process.h> /* For the court of law */
<!-- usw. -->
</pre>

</body>
</html>
```



5.17: Mit CSS lässt sich die Darstellung von präformatiertem Text umfassend beeinflussen.

Das `pre`-Element im Beispiel erhält einen blauen Hintergrund, gelbe Schriftfarbe, und als Schriftart wird »Fixedsys« gewünscht, oder, falls nicht verfügbar, eine Courier-Schrift. Damit der Text nicht so am Rand klebt, wird außerdem ein Innenabstand von 10 Pixeln definiert.

### Weitere Informationen

In der HTML-Referenz im Anhang finden Sie Angaben darüber, wo präformatierter Text vorkommen darf, welche anderen Elemente er enthalten darf, welche Attribute das `pre`-Element haben kann und was bei den einzelnen Attributen zu beachten ist.

## 5.7 Logische Auszeichnungen im Text

In HTML gibt es logische und physische Elemente zur Auszeichnung von Text. Logische Textauszeichnungen haben Bedeutungen wie »betont« oder »emphatisch«. Bei logischen Elementen entscheidet der Web-Browser, wie ein solcher Text hervorgehoben wird (z.B. fett, kursiv oder andersfarbig). In Verbindung mit CSS Stylesheets können Sie logische Textauszeichnungen allerdings nach Wunsch formatieren.

Im Unterschied zu Elementen wie Überschriften, Textabsätzen, Listen usw., die ja auch der logischen Strukturierung von Text dienen, sind die hier beschriebenen Elemente jedoch so genannte Inline-Elemente, während Überschriften, Textabsätze, Listen usw. als Block-Elemente bezeichnet werden. Inline-Elemente erzeugen keinen Absatz (genauer: keine neue Zeile) im Textfluss.

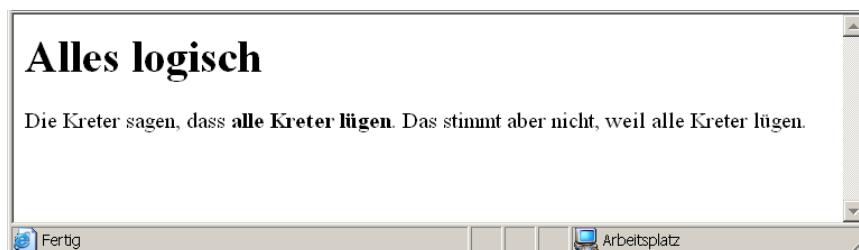
### 5.7.1 HTML-Elemente für logische Auszeichnung im Text

Es stehen verschiedene HTML-Elemente zur Verfügung, um Text logisch auszuzeichnen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Alles logisch</h1>

<p>Die Kreter sagen, da&szlig;
<strong>alle Kreter l&uuml;l;gen</strong>.
Das stimmt aber nicht, weil alle Kreter l&uuml;l;gen.</p>

</body>
</html>
```



5.18: Mit der logischen Textauszeichnung heben Sie zum Beispiel Wörter in einem Satz besonders hervor.

Inline-Elemente für Auszeichnungen im Text müssen – zumindest in der HTML-Variante »Strict« – innerhalb anderer Block-Elemente vorkommen. Im obigen Beispiel ist ein Textabsatz notiert und innerhalb davon eine logische Textauszeichnung. Am Anfang des Textbereichs, der ausgezeichnet werden soll, wird ein einleitendes Tag (im Beispiel das Tag `<strong>`) eingefügt. Am Ende des gewünschten Textbereichs wird ein entsprechendes Abschluss-Tag eingefügt (im Beispiel `</strong>`).

Folgende Elemente dieser Art stehen zur Verfügung:

| Element (Notation)                                | Status | Bedeutung                                                                                      |
|---------------------------------------------------|--------|------------------------------------------------------------------------------------------------|
| <code>&lt;em&gt;...&lt;/em&gt;</code>             |        | Zeichnet einen Text aus mit der Bedeutung »emphatisch«.                                        |
| <code>&lt;strong&gt;...&lt;/strong&gt;</code>     |        | Zeichnet einen Text aus mit der Bedeutung »stark betont«.                                      |
| <code>&lt;code&gt;...&lt;/code&gt;</code>         |        | Zeichnet einen Text aus mit der Bedeutung »dies ist Quelltext«                                 |
| <code>&lt;samp&gt;...&lt;/samp&gt;</code>         |        | Zeichnet einen Text aus mit der Bedeutung »Dies ist ein Beispiel«                              |
| <code>&lt;kbd&gt;...&lt;/kbd&gt;</code>           |        | Zeichnet einen Text aus mit der Bedeutung »dies stellt eine Tastatureingabe dar«               |
| <code>&lt;var&gt;...&lt;/var&gt;</code>           |        | Zeichnet einen Text aus mit der Bedeutung »dies ist eine Variable oder ein variabler Name«     |
| <code>&lt;cite&gt;...&lt;/cite&gt;</code>         |        | Zeichnet einen Text aus mit der Bedeutung »dies ist ein Zitat von einer anderen Quelle«        |
| <code>&lt;dfn&gt;...&lt;/dfn&gt;</code>           |        | Zeichnet einen Text aus mit der Bedeutung »dies ist eine Definition«.                          |
| <code>&lt;acronym&gt;...&lt;/acronym&gt;</code>   |        | Zeichnet einen Text aus mit der Bedeutung »dies ist eine Abkürzung« (z.B. »z.B.«)              |
| <code>&lt;abbr&gt;...&lt;/abbr&gt;</code>         |        | Zeichnet einen Text aus mit der Bedeutung »dies ist eine abgekürzte Schreibweise« (z.B. »WWW«) |
| <code>&lt;q cite="Quelle"&gt;...&lt;/q&gt;</code> |        | Zeichnet einen Text aus mit der Bedeutung »dies ist ein Zitat mit Quellenangabe«               |

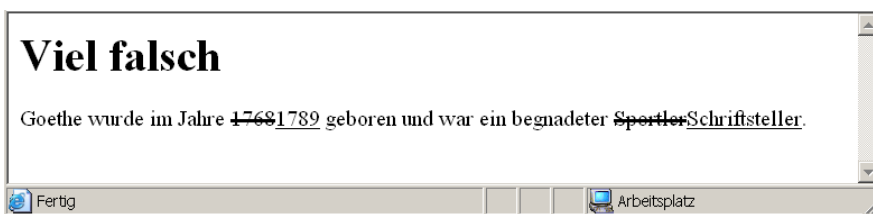
### 5.7.2 Änderungsmarkierungen im Dokument

Wenn Sie Dokumente in HTML bearbeiten, bei denen es wichtig ist, Änderungen zu protokollieren, dann können Sie logische Textauszeichnungen für gelöschte und eingefügte Textpassagen verwenden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Viel falsch</h1>

<p>Goethe wurde im Jahre <del>1768</del><ins>1789</ins>
geboren und war ein begnadeter <del>Sportler</del><ins>Schriftsteller</ins>.</p>

</body>
</html>
```



5.19: Änderungen im Text markieren Sie, damit sie später nachvollziehbar sind.

Mit `<del>...</del>` zeichnen Sie Text aus, der nicht mehr gültig ist (*del* = *delete* = *löschen*). Mit `<ins>...</ins>` können Sie Text auszeichnen, der neu dazugekommen ist (*ins* = *insert* = *einfügen*). Zur exakteren Auszeichnung sind noch zwei Attribute in beiden einleitenden Tags erlaubt:

- Mit dem Attribut `datetime=` im einleitenden Tag von `<ins>` oder `<del>` können Sie den Zeitpunkt der Änderung festhalten. Eine Angabe wie `<ins datetime="2001-06-25T10:59+01:00">` bedeutet: »eingefügt am 25.6.2001 um 10 Uhr 59« nach Zeitzone UTC-Zeit plus 1 Stunde. Die Uhrzeit/Zeitzoneangabe können Sie auch weglassen, etwa so: `<ins datetime="2001-06-25">`. UTC bedeutet »Coordinated Universal Time«.
- Mit dem Attribut `cite=` können Sie einen URI angeben, der als der Grund für die Änderung gelten kann – beispielsweise ein verbindliches Dokument, auf dem Ihr Dokument beruht und das mittlerweile geändert wurde. Eine Angabe wie `<ins cite="http://www.w3.org/TR/1999/REC-html401-19991224">` bedeutet: Dieser Text wurde aufgrund der HTML-Spezifikation 4.01 vom 24.12.1999 eingefügt (der angegebene URI ist die Adresse, an der die HTML-Spezifikation zu finden ist).



**Beachten Sie:** Die beiden Elemente `ins` und `del` haben eine Sonderstellung, was ihr Vorkommen betrifft. Diese beiden Elemente dürfen entweder innerhalb von Block-Elementen vorkommen – dann fungieren sie selber als Inline-Elemente. Oder sie werden außerhalb von Block-Elementen notiert – dann fungieren sie selber als Block-Elemente und können mehrere andere Block-Elemente und deren gesamten Inhalt einschließen.

### 5.7.3 Textrichtung ändern

Wenn Sie HTML in Verbindung mit Sprachen benutzen, die eine andere Textrichtung als von links nach rechts haben, kann es zu Konflikten bei der Textrichtung kommen, vor allem wenn Sie zwei Sprachen mit unterschiedlicher Textrichtung im Text gemischt verwenden. Wenn Sie Unicodes in HTML notieren, wird dabei normalerweise automatisch auch die Textrichtung der Sprache berücksichtigt. Wenn Sie etwa die Unicodes für ein arabisches Wort von links nach rechts notieren, sollte die Software die Zeichenkette dennoch so auflösen, dass die Darstellung von rechts nach links erfolgt, weil das die Textrichtung im Arabischen ist. Sollte das aus irgendwelchen Gründen nicht funktionieren, können Sie mit Hilfe eines HTML-Elements nachhelfen.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Umgekehrt</h1>

<p>Hier wird das Wort &quot;umgekehrt&quot; einfach <bdo dir="rtl">umgekehrt</bdo>.

</body>
</html>
```



5.20: Je nach den verwendeten Schriftzeichen macht es mehr oder weniger Sinn, die Schreibrichtung umzukehren.

Mit `<bdo dir="rtl">` leiten Sie einen Text ein, der von rechts nach links dargestellt wird, und mit `<bdo dir="ltr">` einen Text, der von links nach rechts dargestellt wird. `</bdo>` beendet den Text mit der anderen Schreibrichtung (*bdo* = *bidirectional overwrite* = *Überschreiben der automatischen Bidirektionalität*). Mit dem Attribut `dir=` erzwingen Sie die Textrichtung (*rtl* = *right to left* = *rechts nach links*).

Das Attribut `dir=` gehört zu den Universalattributen und darf in allen einleitenden HTML-Tags stehen, in denen Text vorkommen kann. Das `bdo`-Element ist für solche Fälle gedacht, in denen `dir=` in anderen HTML-Elementen zu Problemen führt, beispielsweise bei verschachtelten Tags mit widersprüchlichen Anweisungen zur Textrichtung.

### 5.7.4 Logische Elemente für Textauszeichnung mit CSS formatieren

Wie logische Elemente zur Textauszeichnung genau dargestellt werden, darauf haben Sie mit HTML keinen Einfluss. Die Browser benutzen Default-Formatierungen. Mit CSS Stylesheets können Sie solche Elemente jedoch nach Wunsch formatieren. Bei Verwendung von CSS Stylesheets lesen Sie zunächst in Kapitel 15 nach, wie man CSS-Formate definieren kann. Anschließend sind Sie in der Lage, CSS-Eigenschaften anzuwenden. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Schriftformatierung, Hintergrundfarben und -bilder. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Viele viele bunte Logi's</h1>

<p>
Text mit dem <em style="color:blue">em-Element</em><br>
Text mit dem <strong style="color:red">strong-Element</strong><br>
Text mit dem <code style="color:purple">code-Element</code><br>
<!-- usw. -->
</p>

</body>
</html>
```

Die verschiedenen logischen Elemente im Beispiel erhalten unterschiedliche Schriftfarben. Formatieren können die Browser allerdings nur Elemente, die sie kennen.

## 5.8 Physische Auszeichnungen im Text

In HTML gibt es physische und logische Elemente zur Auszeichnung von Text. Physische Textauszeichnungen haben Bedeutungen wie »fett« oder »kursiv«, stellen also direkte Angaben zur gewünschten Schriftformatierung dar. Bei physischen Elementen sollte der Web-Browser eine Möglichkeit finden, den so ausgezeichneten Text entsprechend darzustellen. Ebenso wie die

logischen Elemente zur Auszeichnung von Text gehören die hier beschriebenen Elemente zu den Inline-Elementen.

### 5.8.1 HTML-Elemente für physische Auszeichnung im Text

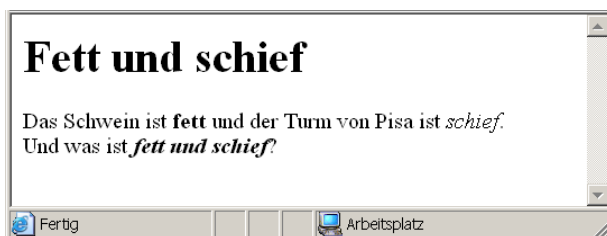
Es stehen verschiedene HTML-Elemente zur Verfügung, um Text physisch auszuzeichnen. Inline-Elemente für Auszeichnungen im Text müssen – zumindest in der HTML-Variante »Strict« – innerhalb anderer Block-Elemente vorkommen. Im nachfolgenden Beispiel ist ein Textabsatz notiert und innerhalb davon stehen mehrere physische Textauszeichnungen. Am Anfang des Textbereichs, der ausgezeichnet werden soll, wird ein einleitendes Tag (im Beispiel die Tag `<b>` und `<i>`) eingefügt. Am Ende des gewünschten Textbereichs wird ein entsprechendes Abschluss-Tag eingefügt (z.B. `</b>` bzw. `</i>`).

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Fett und schief</h1>

<p>Das Schwein ist <b>fett</b> und der Turm von Pisa ist <i>schief</i>.<br>
Und was ist <b><i>fett und schief</i></b>?</p>

</body>
</html>
```



5.21: Physische Textauszeichnungen lassen sich auch kombinieren.

Folgende Elemente dieser Art stehen zur Verfügung:

| Element (Notation)                            | Status                                                                                                                                                                                                                                                                                                                                  | Bedeutung                                                                                                     |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>&lt;b&gt;...&lt;/b&gt;</code>           |     | Zeichnet einen Text als fett aus.                                                                             |
| <code>&lt;i&gt;...&lt;/i&gt;</code>           |     | Zeichnet einen Text als kursiv aus.                                                                           |
| <code>&lt;tt&gt;...&lt;/tt&gt;</code>         |     | Zeichnet einen Text als dicktengleich formatiert aus ( <i>tt</i> = <i>Teletyper</i> = <i>Fernschreiber</i> ). |
| <code>&lt;u&gt;...&lt;/u&gt;</code>           |     | Zeichnet einen Text als unterstrichen aus.                                                                    |
| <code>&lt;strike&gt;...&lt;/strike&gt;</code> |     | Zeichnet einen Text als durchgestrichen aus.                                                                  |
| <code>&lt;s&gt;...&lt;/s&gt;</code>           |     | Zeichnet einen Text als durchgestrichen aus.                                                                  |
| <code>&lt;big&gt;...&lt;/big&gt;</code>       |     | Zeichnet einen Text größer als normal aus.                                                                    |
| <code>&lt;small&gt;...&lt;/small&gt;</code>   |     | Zeichnet einen Text kleiner als normal aus.                                                                   |
| <code>&lt;sup&gt;...&lt;/sup&gt;</code>       |     | Zeichnet einen Text als hochgestellt aus.                                                                     |
| <code>&lt;sub&gt;...&lt;/sub&gt;</code>       |     | Zeichnet einen Text als tiefgestellt aus.                                                                     |

**Beachten Sie:** Sie können auch – wie im obigen Beispiel – Elemente dieser Art verschachteln. Dadurch sollten sich die Effekte addieren. Ein Text, der also beispielsweise als fett und innerhalb davon als kursiv ausgezeichnet wird, sollte fettkursiv dargestellt werden.

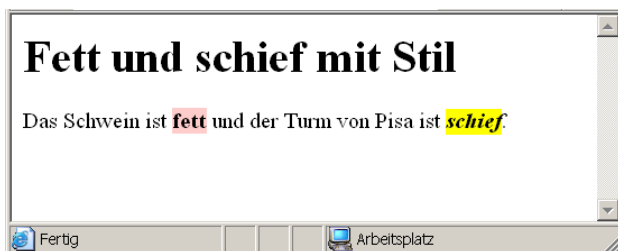
## 5.8.2 Physische Elemente für Textauszeichnung zusätzlich mit CSS formatieren

Mit CSS Stylesheets können Sie auch physische Elemente nach weiteren Wünschen formatieren. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Schriftformatierung, Hintergrundfarben und -bilder. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Fett und schief mit Stil</h1>

<p>Das Schwein ist <b style="background-color:#FFCCCC">fett</b> und der Turm
von Pisa ist <i style="font-weight:bold; background-color:#FFFF00">schief</i>.</p>

</body>
</html>
```



5.22: Mit CSS erweitern Sie die Darstellung von Text um vielfältige Optionen wie etwa eine farbige Hinterlegung.

Die beiden Elemente für fette und kursive Schriftauszeichnung erhalten im Beispiel zusätzlich unterschiedliche Hintergrundfarben. Das `i`-Element wird außerdem fett ausgezeichnet – mit Hilfe von CSS.

## 5.9 Allgemeine Elemente für Textbereiche

### 5.9.1 Allgemeines Block-Element

Sie können mehrere Elemente wie Text, Grafiken, Tabellen usw. in einen gemeinsamen Bereich einschließen. Dieses allgemeine Element bewirkt nichts weiter, als dass es in einer neuen Zeile des Fließtextes beginnt. Ansonsten hat es keine Eigenschaften. Es ist dazu gedacht, um mit Hilfe von CSS formatiert zu werden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<div align="center">
<h1>Alles zentriert</h1>
<ul>
<li>alles zentriert</li>
<li>alles?</li>
<li>alles!</li>
</ul>
</div>

<div align="right">
<h1>Alles rechts am Rand</h1>
<ul>
```

```
<li>alles rechts am Rand</li>
<li>alles?</li>
<li>alles!</li>
</ul>
</div>

</body>
</html>
```

Mit `<div>` leiten Sie ein allgemeines Blockelement ein, in das Sie mehrere andere Block-Elemente einschließen können (*div = division = Bereich*). Alles, was zwischen diesem Tag und dem abschließenden `</div>` steht, wird als Teil des Bereichs interpretiert.

Einen solchen Bereich und alle seine enthaltenen Elemente können Sie mit dem Attribut `align=` im einleitenden `<div>`-Tag ausrichten. Mit `align="center"` richten Sie den Bereich mit allen seinen Elementen zentriert aus (*align = Ausrichtung, center = zentriert*), mit `align="right"` rechtsbündig (*right = rechts*). Mit `align="justify"` werden innere Elemente wie freistehender Text, Textabsätze oder Überschriften als Blocksatz ausgerichtet. Mit `align="left"` erzwingen Sie die linksbündige Ausrichtung von Elementen (Voreinstellung).

**Beachten Sie:** Das `div`-Element wird weiterhin zum HTML-Standard gehören. Das Attribut `align=` ist jedoch als *deprecated* eingestuft und soll künftig entfallen. Stattdessen sollten Sie mit CSS Stylesheets arbeiten.

### 5.9.2 Älteres Block-Element für zentrierte Bereiche

Sie können mehrere Absätze, bestehend aus ganz verschiedenen Elementen wie Text, Grafiken, Tabellen usw., gemeinsam zentriert ausrichten. Dieses Element ist jedoch als *deprecated* eingestuft. Es ist mittlerweile eigentlich überflüssig und sollte nur noch in begründeten Ausnahmefällen (z.B. Abwärtskompatibilität zu Netscape 1.1) benutzt werden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<center>
<h1>Zentrierter Kunsttext</h1>
<pre>
<big>.:*A*:_.:*S*:_.:*C*:_.:*I*:_.:*I*:_.:*~*:_.:*A*:_.:*R*:_.:*T*:_.:</big>
</pre>
</center>

</body>
</html>
```

Mit `<center>` leiten Sie einen Bereich ein, in dem alles, was folgt, zentriert ausgerichtet wird (*center* = *zentriert*). Das können z.B. Überschriften, Text, Grafiken oder Tabellen sein. Mit dem abschließenden Tag `</center>` beenden Sie die Zentrieranweisung.

### 5.9.3 Allgemeines Inline-Element

Analog zum `div`-Element, das andere Block-Elemente enthalten kann, gibt es ein Element, das Text und andere Inline-Elemente enthalten kann, selbst aber keinerlei Eigenschaften hat und nichts bewirkt. Es ist dazu gedacht, um mit Hilfe von CSS formatiert zu werden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1 style="font-size:500%">
<span style="color:blue">A</span>
<span style="color:red">B</span>
<span style="color:green">C</span>
</h1>

</body>
</html>
```

`<span>` leitet einen allgemeinen Inline-Bereich ein. `</span>` beendet diesen Bereich. Das Beispiel zeigt eine Überschrift 1. Ordnung, die selbst bereits mit CSS formatiert ist und dadurch auf 500% Schriftgröße gebracht wird. Innerhalb der Überschrift sind die drei Buchstaben ABC notiert. Jeder davon ist in ein `span`-Element eingeschlossen, und bei jedem `span`-Element wird mit Hilfe von CSS eine andere Schriftfarbe zugewiesen.

**Beachten Sie:** In der »Strict«-Variante dürfen Inline-Elemente nur innerhalb von Block-Elementen notiert werden. Im obigen Beispiel ist das der Fall, da die `span`-Elemente innerhalb eines Überschriftenelements vorkommen.

### 5.9.4 Allgemeine Elemente formatieren mit CSS

Allgemeine Elemente für Textbereiche erwachen durch Formatierung erst zum Leben. Mit CSS Stylesheets können Sie solche Elemente jedoch nach Wunsch formatieren. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Schriftformatierung, Ausrichtung und Absatzkontrolle, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder, Positionierung und Anzeige von Elementen. Ein Beispiel:

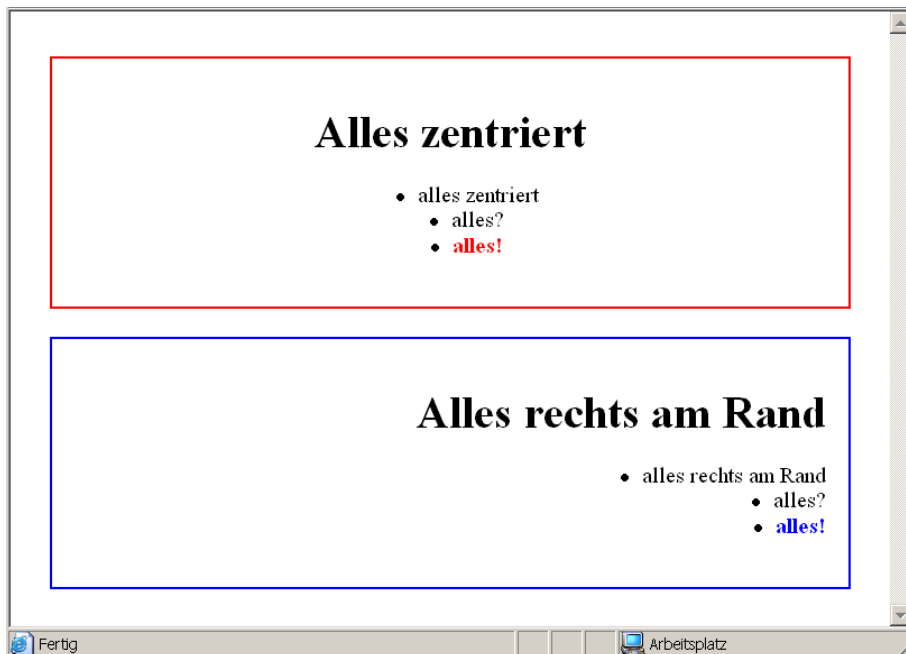
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
```

```
<head>
<title>Text des Titels</title>
</head>
<body>

<div style="text-align:center; padding:20px; border:thin solid red; margin:25px">
<h1>Alles zentriert</h1>
<ul>
<li>alles zentriert</li>
<li>alles?</li>
<li><span style="font-weight:bold; color:red">alles!</span></li>
</ul>
</div>

<div style="text-align:right; padding:20px; border:thin solid blue; margin:25px">
<h1>Alles rechts am Rand</h1>
<ul>
<li>alles rechts am Rand</li>
<li>alles?</li>
<li><span style="font-weight:bold; color:blue">alles!</span></li>
</ul>
</div>

</body>
</html>
```



5.23: Über CSS ziehen Sie zum Beispiel Rahmen um einen Bereich oder ändern die Ausrichtung.



Das Beispiel bewirkt eine ähnliche Ausgabe wie das im Abschnitt *Allgemeines Block-Element*, diesmal jedoch mit CSS Stylesheets formatiert und damit auch geeignet für die HTML-Variante »Strict«. Beide `div`-Bereiche erhalten einen dünnen durchgezogenen Rahmen, der erste rot, der zweite blau. Ein Innenabstand von 20 Pixeln sorgt dafür, dass der Text nicht am Rahmen klebt, und ein Außenabstand von 25 Pixeln stellt sicher, dass links, rechts über und unter den Rahmen Platz entsteht. Der letzte Listenpunkt in beiden Aufzählungslisten wird jeweils mit dem leeren Inline-Element `span` ausgezeichnet und erhält jeweils eine fette Schrift und eine Schriftfarbe zugewiesen.

## 5.10 Trennlinien

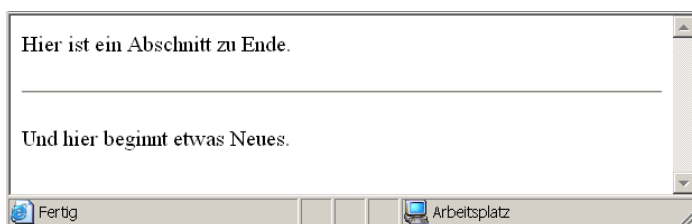
### 5.10.1 Trennlinien definieren

Trennlinien dienen der optischen Abgrenzung von nicht unmittelbar zusammengehörigen Textabschnitten oder allgemein zur Auflockerung. Eine Trennlinie erzeugt einen eigenen Absatz. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<p>Hier ist ein Abschnitt zu Ende.</p>
<hr>
<p>Und hier beginnt etwas Neues.</p>

</body>
</html>
```



5.24: Mit Trennlinien gliedern Sie Texte und andere Elemente in überschaubare Einheiten

`<hr>` fügt eine Trennlinie ein (*hr* = *horizontal rule* = *Querlinie*). Dabei ist es egal, ob das Tag am Ende der Zeile des vorherigen Absatzes steht oder in einer eigenen Zeile (wie im Beispiel), oder am Anfang des folgenden Absatzes.

**Beachten Sie:** Wenn Sie XHTML-konform arbeiten, müssen Sie das `hr`-Element als inhaltsleer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<hr />`. Weitere Informationen dazu im Kapitel 14.1.11 zu *XHTML und HTML*.

### 5.10.2 Trennlinien mit HTML gestalten

Mit Hilfe diverser HTML-Attribute im `<hr>`-Tag können Sie eine Trennlinie auffälliger gestalten. Diese Attribute sind allerdings allesamt als *deprecated* eingestuft und sollen künftig aus dem HTML-Standard entfallen. Empfohlen wird die Gestaltung von Trennlinien mit CSS. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<p>Hier ist ein Abschnitt zu Ende.</p>
<hr noshade width="300" size="3" align="left">
<p>Hier beginnt etwas Neues.</p>

</body>
</html>
```

Durch das Attribut `noshade` erreichen Sie, dass der Browser die Trennlinie massiv und durchgezogen, also nicht schattiert anzeigt (*noshade* = *unschattiert*). Wenn Sie XHTML-konform arbeiten, müssen Sie das Attribut in der Form `noshade="noshade"` notieren, da XML-basierte Sprachen keine Attribute ohne Wertzuweisung erlauben.

Durch das Attribut `width=` (*width* = *Breite*) erreichen Sie, dass der Browser die Trennlinie so breit anzeigt wie angegeben. Sie können eine Zahl oder einen Prozentwert angeben. Mit einer Zahl, z.B. 300, erzwingen Sie, dass die Trennlinie so viel Pixel breit dargestellt wird wie angegeben. Mit einem Prozentwert erreichen Sie, dass die Trennlinie maximal so viel Breite des Anzeigefensters einnimmt wie angegeben. Für eine prozentuale Angabe notieren Sie hinter der Zahl einfach ein Prozentzeichen (%).

Durch das Attribut `size=` (*size* = *Größe*) können Sie die Höhe (Dicke) der Trennlinie bestimmen. Die Voreinstellung beträgt 2 Punkt. Mit dem Wert 1 erzwingen Sie also eine besonders dünne Trennlinie, mit Werten über 2 können Sie dickere als die normalen Trennlinien erzeugen.

Mit `align="left"` erreichen Sie, dass die Trennlinie linksbündig ausgerichtet wird (*align* = *Ausrichtung*, *left* = *links*). Mit `align="right"` wird die Trennlinie rechtsbündig ausgerichtet (*right* = *rechts*) und mit `align="center"` zentriert (Voreinstellung). Das Ausrichten von Trennlinien ist allerdings nur in Verbindung mit dem Attribut `width=` sinnvoll, da die Trennlinie sonst stets über die gesamte verfügbare Breite geht.

Der MS Internet Explorer interpretiert darüber hinaus noch das Attribut `color=`. Erlaubt ist dabei eine HTML-übliche Farbangabe. Dieses Attribut hat jedoch nie zum HTML-Standard gehört.

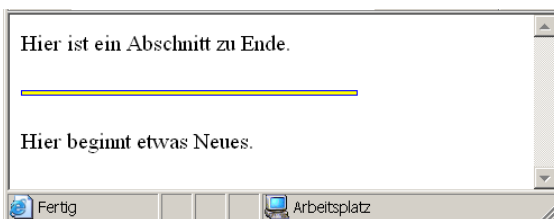
### 5.10.3 Trennlinien mit CSS gestalten

Das `<hr>`-Tag können Sie auch mit CSS gestalten. Das ist strikt HTML-konform und Sie haben noch deutlich mehr Gestaltungsmöglichkeiten als mit HTML-Attributen. Leider interpretiert jedoch der Netscape-Browser CSS-Eigenschaften in Verbindung mit Trennlinien noch nicht (Stand: Version 6.1), und auch der Internet Explorer noch nicht so lange. Die Verwendung ist also mit Vorsicht zu genießen. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Ausrichtung und Absatzkontrolle, Außenrand und Abstand, Rahmen, Positionierung und Anzeige von Elementen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<p>Hier ist ein Abschnitt zu Ende.</p>
<hr noshade style="width:300px; color:yellow; height:5px; text-align:left;
border:1px solid blue;">
<p>Hier beginnt etwas Neues.</p>

</body>
</html>
```



5.25: Mit CSS lassen sich Trennlinien weitgehend beeinflussen – allerdings ist der Effekt nur in den neuesten Browser-Versionen sichtbar.

Das `hr`-Element erhält im Beispiel eine Breite von 300 Pixeln, eine Höhe von 5 Pixeln, wird linksbündig ausgerichtet, wird in gelber Farbe dargestellt und erhält noch einen dünnen blauen Rahmen drum herum.

## 5.11 Ältere Elemente zur Schriftformatierung

### 5.11.1 Schriftart, Schriftgröße und Schriftfarbe

Die hier beschriebenen HTML-Elemente sind als *deprecated* eingestuft, das heißt, sie sollen in Zukunft aus dem HTML-Standard entfernt werden. Da es sich bei diesen Elementen um die schlimmste »Verunreinigung« von HTML als reiner Struktursprache handelt, die bisher passiert ist, gibt es richtige Kampfschriften und Hetzkampagnen gegen die weitere Verwendung dieser Elemente im Web. Diese Elemente stellten eine Notlösung in jener Zeit dar, als HTML noch keine Formattersprache wie CSS zur Seite stand.

Im Hinblick auf die Rückwärtskompatibilität zu Netscape 3.x kann der Gebrauch der hier beschriebenen Elemente noch nicht grundsätzlich verdammt werden. Aber bevor Sie auf Style-Angaben verzichten, verzichten Sie mittlerweile lieber auf die hier beschriebenen Elemente und benutzen Sie die CSS-Entsprechungen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<p>
<font size="7">Ziemlich riesiger Text</font><br>
<font size="1">Ziemlich winziger Text</font><br>
<font size="+1">Text etwas grö&ouml;r als normal</font><br>
<font size="-2">Text deutlich kleiner als normal</font>
</p>

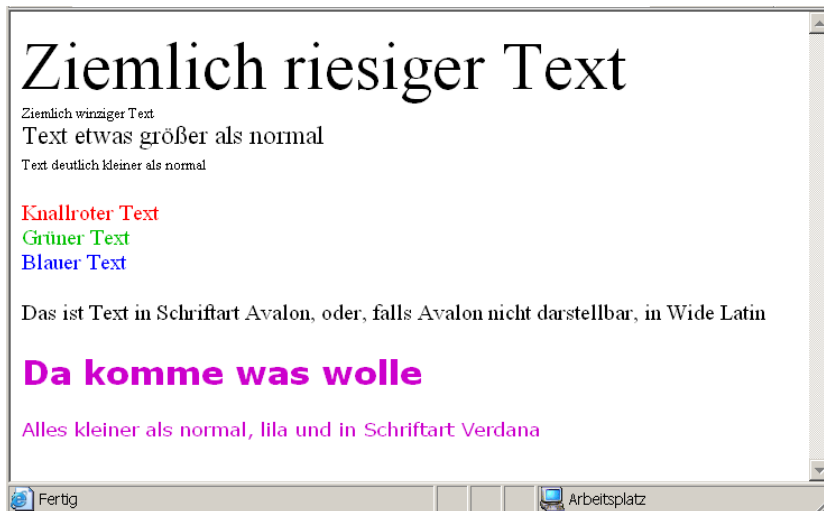
<p>
<font color="#FF0000">Knallroter Text</font><br>
<font color="#00C000">Gr&uuml;ner Text</font><br>
<font color="#0000FF">Blauer Text</font>
</p>

<p>
<font face="Avalon,Wide Latin">Das ist Text in Schriftart Avalon,
oder, falls Avalon nicht darstellbar, in Wide Latin</font>
</p>

<basefont size="2" color="#CC00CC" face="Verdana">

<h2>Da komme was wolle</h2>
<p>
Alles kleiner als normal, lila und in Schriftart Verdana
```

```
</p>
</body>
</html>
```



5.26: Mit den ausgemusterten Tags zur Schriftformatierung lässt sich eine Menge erreichen.

`<font>...</font>` definieren Sie einen Bereich für Schriftformatierung. Das Element darf selber nur Inline-Elemente enthalten, keine Block-Elemente. Wenn Sie also beispielsweise Textabsätze damit auszeichnen wollen, müssen Sie das `font`-Element jedes Mal innerhalb von `<p>...</p>` notieren. Die Browser sehen es zwar nicht so eng, wenn Sie es schlampiger verwenden, aber dann ist es nicht mehr HTML-konform. Durch Attribute im einleitenden `<font>`-Tag können Sie Schrift formatieren.

- `size=` bestimmt die Schriftgröße (*font size = Schriftgröße*). Sie können den Wert in Zahlen zwischen 1 und 7 angeben, oder relativ im Verhältnis zur Normalschriftgröße mit `+Zahl` bzw. `-Zahl`. Die Normalschriftgröße ist 3. Mit `</font>` beenden Sie den Abschnitt mit anderer Schriftgröße. Bei den Angaben zu `<font size=...>` handelt es sich um relative Werte. Wenn der Anwender eine 12-Punkt-Schrift eingestellt hat, hat beispielsweise `<font size=5>` eine andere Wirkung, als wenn der Anwender eine 9-Punkt-Schrift eingestellt hat. Die Schriftgröße, die Sie mit `<font size=...>` bestimmen können, ist auch relativ zu der Schriftgröße des Absatztyps.
- `<font color=...>` bestimmt die Schriftfarbe (*font color = Schriftfarbe*). Erlaubt ist eine HTML-übliche Farbangabe.
- `<font face=...>` bestimmt die Schriftart (*font face = Schriftart*). Als Wert können Sie die Namen einer oder mehrerer Schriftarten zuweisen. Wenn Sie mehrere Schriftarten angeben, trennen Sie die Schriftartnamen durch Kommata. Bei mehreren angegebenen Schriftarten versucht der WWW-Browser zuerst, den Text in der ersten angegebenen Schriftart darzustellen (im Beispiel in *Avalon*). Wenn diese Schrift beim Anwender nicht installiert ist, wird versucht, die zweite angegebene Schrift (im Beispiel *Wide Latin*) zu nehmen. Wenn keine der

angegebenen Schriftarten beim Anwender darstellbar ist, bleibt die Angabe wirkungslos, d.h. der Text wird in der vom Anwender eingestellten Schrift angezeigt. Verwenden Sie exakte Schriftartnamen. Unter MS-Windows können Sie eingestellte Schriftarten und ihre genauen Namen beispielsweise über die Systemsteuerung ermitteln.

- Das zweite Element zur Schriftformatierung ist das `basefont`-Element. Es ist ein allein stehendes Tag und bewirkt, dass die darin definierten Formatierungen für alle weiteren Elemente in der Datei gelten, solange, bis ein neues `basefont`-Element notiert ist oder bis zum Ende des Dokuments. Attribute und Werte sind die gleichen wie beim `font`-Element.

**Beachten Sie:** Das `font`-Element ist HTML 3.2, `basefont` dagegen HTML 4.0. Im einleitenden `<font>`-Tag können Sie ebenso wie im `<basefont>`-Tag mehrere Attribute notieren, also `size=`, `color=` und `face=` mischen. Netscape 4.x interpretiert `<basefont color=` und `<basefont face=` nicht. Wenn Sie XHTML-konform arbeiten, müssen Sie das `basefont`-Element als inhalts-leer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<basefont size="2" />`. Weitere Informationen dazu im Kapitel 14.1.11 über *XHTML und HTML*.

### 5.11.2 CSS-Entsprechungen

Mit CSS Stylesheets können Sie alle Block- und Inline-Elemente nach Wunsch formatieren. Maßgeblich sind im hier beschriebenen Zusammenhang die CSS-Eigenschaften zur Schriftformatierung. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Schriftart, Schriftgröße und Schriftfarbe - CSS-
Entsprechungen</title>
</head>
<body>

<p>
<span style="font-size:250%">Ziemlich riesiger Text</span><br>
<span style="font-size:50%">Ziemlich winziger Text</span><br>
<span style="font-size:1.2em">Text etwas größer als normal</span><br>
<span style="font-size:0.8em">Text deutlich kleiner als normal</span>
</p>

<p>
<span style="color:#FF0000">Knallroter Text</span><br>
<span style="color:#00C000">Grüner Text</span><br>
<span style="color:#0000FF">Blauer Text</span>
</p>

<p>
<span style="font-family:Avalon,Wide Latin">Das ist Text in Schriftart Avalon,
oder, falls Avalon nicht darstellbar, in Wide Latin</span>
```

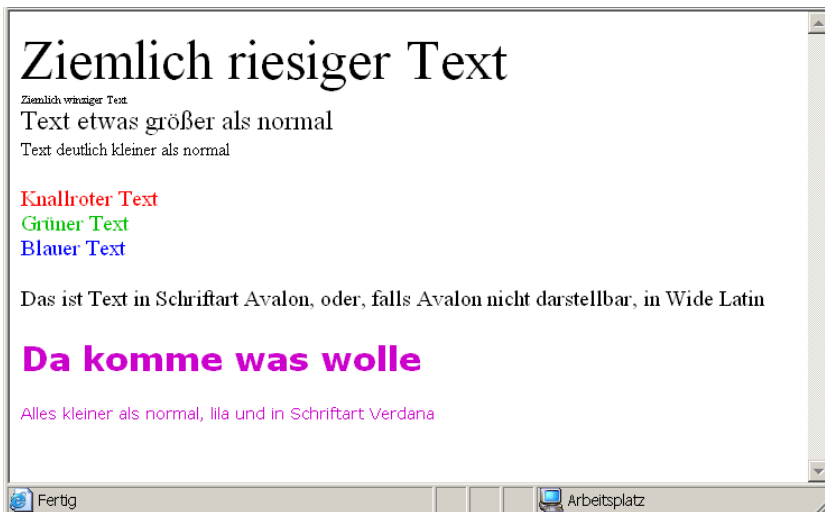
```

</p>

<div style="font-size:70%; color:#CC00CC; font-family:Verdana">
<h2>Da komme was wolle</h2>
<p>
Alles kleiner als normal, lila und in Schriftart Verdana
</p>
</div>

</body>
</html>

```



5.27: Mit CSS lassen sich weitgehend die gleichen Effekte (und noch viel mehr) wie mit den ausgemusterten Tags erreichen.

Das Beispiel bewirkt in etwa die gleiche Ausgabe wie das Beispiel zuvor, jedoch ohne die Elemente `font` und `basefont`. Für die CSS-Schriftformatierungen innerhalb der Textabsätze wird das allgemeine Inline-Element `span` eingesetzt. Anstelle des allein stehenden `<basefont>`-Tags wird der restliche Abschnitt in ein allgemeines Block-Element `div` eingeschlossen, das die CSS-Formatierungen zugewiesen bekommt.

## 5.12 Ältere Browser-proprietäre Elemente

### 5.12.1 Lauftext – Marquee (Microsoft)

Sie können erreichen, dass ein Text mitten im übrigen Text automatisch horizontal scrollt, so wie in dem von Windows bekannten Bildschirmschoner »Marquee«. Diese Angaben gehören nicht zum HTML-Standard und werden nur vom Internet Explorer interpretiert. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<marquee scrollamount="50" scrolldelay="5">
<b>Dieser Text wird ziemlich schnell bewegt</b>
</marquee>
<marquee behavior="alternate">
<b>Dieser Text wird hin und her bewegt</b>
</marquee>
</body>
</html>
```

Mit `<marquee>` leiten Sie den zu scrollenden Inhalt ein. Dahinter folgt der Inhalt, der gescrollt werden soll. Am Ende notieren Sie ein abschließendes `</marquee>`.

- Mit `direction="right"` im einleitenden Tag erzwingen Sie die Laufrichtung von links nach rechts (*direction* = *Richtung*, *right* = *rechts*). Mit `direction="left"` stellen Sie die Laufrichtung auf rechts nach links (Voreinstellung).
- Durch das Attribut `scrollamount=` bestimmen Sie die Pixelanzahl zwischen den Scroll-Zuständen. Mit `scrolldelay=` legen Sie die Verzögerung zwischen zwei Scroll-Zuständen in Millisekunden fest (*amount* = *Zustand*, *delay* = *Verzögerung*).
- Mit `behavior="alternate"` erzwingen Sie den Wechsel-Effekt (*behavior* = *Verhalten*, *alternate* = *wechselnd*).
- Durch das Attribut `bgcolor=` erzwingen Sie, dass der gescrollte Text in einem andersfarbigen Rahmen erscheint (*bgcolor* = *background color* = *Hintergrundfarbe*). Für die Farbauswahl gelten die Regeln zum Definieren von Farben in HTML.
- Mit `align="top"` erreichen Sie, dass der umgebende Text obenbündig zu dem Marquee-Text ausgerichtet wird. Mit `align="middle"` wird der umgebende Text mittig ausgerichtet, mit `align="bottom"` untenbündig (*align* = *Ausrichtung*, *top* = *oben*, *middle* = *mittig*, *bottom* = *unten*).
- Durch `height=` bestimmen Sie die Höhe des Marquee-Bereichs (*height* = *Höhe*). Diese Angabe ist vor allem interessant, wenn Sie für den Marquee-Bereich eine eigene Hintergrundfarbe bestimmen. Mit der Höhe des Marquee-Bereichs legen Sie dann die Höhe des andersfarbigen Kastens fest, in dem der Marquee-Text gescrollt wird. Die Angabe erfolgt entweder absolut in Pixeln oder in Prozent relativ zur Höhe des Anzeigefensters.
- Mit `hspace=` bestimmen Sie den Abstand zwischen Marquee-Text und dem Text davor bzw. dahinter (*hspace* = *horizontal space* = *horizontaler Abstand*). Dieses Attribut ist nur von Bedeutung, wenn Sie den Marquee-Bereich mitten im Text definieren. Der Wert erfolgt in Pixeln. Der angegebene Abstand gilt sowohl für den Text davor als auch für den Text dahinter.

**Beachten Sie:** Viele Leute wollen ja unbedingt so einen Lauftext auf ihrer Seite haben und sind ganz traurig, dass andere Browser das `marquee`-Element nicht kennen. Es gibt mehrere Alterna-



tiven zum `marquee`-Element. Eine davon ist, mit Hilfe von Dynamischem HTML einen Lauf-text-Effekt zu programmieren (Kapitel 21). Eine andere ist, so etwas mit Java oder Flash zu tun. Auf der Buch-CD finden Sie ein Beispiel mit einem Java-Applet.

### 5.12.2 Mehrspaltiger Textfluss (Netscape)

Netscape unterstützt in den Versionen 3.x und 4.x (nicht mehr in 6.x!) eine Methode, mehrspaltigen Fließtext im Zeitungsstil in HTML zu realisieren. Dieses Feature gehört jedoch nicht zum HTML-Standard. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<multicol cols="3" gutter="10">
<p>Lesen des folgenden Textes ... usw.</p>
</multicol>

</body>
</html>
```



5.28: Das Tag für mehrspaltigen Text funktioniert nur bei Netscape 3.x und 4.x.

Mit `<multicol>` leiten Sie einen mehrspaltigen Abschnitt ein (*multicol* = *multiple columns* = *mehrere Spalten*). Zwingend erforderlich ist das Attribut `cols=`, mit dem Sie die Anzahl der ge-

wünschten Spalten festlegen. Daran anschließend können beliebige andere Elemente folgen. Mit `</multicol>` wird der mehrspaltige Abschnitt beendet.

Alles, was zwischen dem einleitenden Tag `<multicol>` und dem abschließenden `</multicol>` steht, wird automatisch so verteilt, dass es sich optimal über die angegebene Anzahl Spalten erstreckt. Bei Bildern, Java-Applets usw. kann es allerdings zu unschönen Anzeigen kommen. Denn anders als bei Tabellen richtet sich die Breite einer Spalte nicht nach dem breitesten Inhalt, sondern nach der rechnerischen Spaltenbreite. Das kann dazu führen, dass Grafiken, die breiter sind als die Spalte, in die Nachbarspalte hineinreichen und sich mit dem dort stehenden Inhalt überlappen.

Durch das Attribut `width=` können Sie die Gesamtbreite der Spalten beeinflussen. Sie können eine Zahl oder einen Prozentwert angeben. Mit einer Zahl, z.B. 500, erzwingen Sie, dass alle Spalten zusammengenommen so viel Pixel breit sind wie angegeben. Mit einem Prozentwert erreichen Sie, dass alle Spalten zusammengenommen maximal so viel Breite des Anzeigefensters einnehmen wie angegeben. Für eine prozentuale Angabe notieren Sie hinter der Zahl einfach ein Prozentzeichen.

Mit dem Attribut `gutter=` können Sie den Abstand zwischen den Spalten vergrößern oder verkleinern. Die Voreinstellung ist ein Abstand von 10 Pixeln. Mit einem höheren Wert erreichen Sie einen größeren Spaltenabstand, mit einem niedrigeren Wert einen geringeren Spaltenabstand.

**Beachten Sie:** Eine Alternative zu diesem HTML-Element gibt es derzeit nicht. In CSS war mehrspaltiger Textfluss einmal vorgesehen, wurde dann jedoch wieder verworfen. Mehrere Spalten nebeneinander können Sie durch Tabellen ohne sichtbaren Rahmen erzwingen (Kapitel 8.6). Dabei bricht jedoch der Text nicht automatisch zwischen den Spalten um.

### 5.12.3 Raumschaffer (Netscape)

Ein HTML-Element, das von Netscape 3.x und 4.x (nicht mehr 6.x!) interpretiert wird, erlaubt es, horizontale und vertikale Leerräume zu schaffen. Dadurch lassen sich beispielsweise Absatzabstände vergrößern. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<p>Wer nichts wagt, kann nichts verlieren.</p>
<spacer type="vertical" size="50">
<p>Wer sich nicht sicher ist, kann sich nicht irren.</p>

</body>
</html>
```

Mit `<spacer>` leiten Sie einen Zwischenraum ein (*spacer* = *Raumschaffer*). Notieren Sie das Tag genau an der Stelle, wo Sie den Zwischenraum wünschen. Mit `type="vertical"` geben Sie an, dass es ein vertikaler Zwischenraum sein soll, wie es für Absatzabstände erforderlich ist. Mit

`type="horizontal"` können Sie auch horizontale Abstände erzeugen. Mit dem Attribut `size=` bestimmen Sie die Größe des Abstands. Die Angabe erfolgt in Pixeln (*size=Größe*).

Durch `type="block"` (*block = Rechteck*), `width=` (*width=Breite*) und `height=` (*height=Höhe*) definieren Sie einen rechteckigen Leerraum. Mit `width=` bestimmen Sie die Breite des Rechtecks, mit `height=` die Höhe. Beide Angaben erfolgen in Pixeln. Wenn Text um das Rechteck fließen soll, sollten Sie die Ausrichtung festlegen. Mit der Angabe `align="left"` fließt der folgende Text rechts um das links ausgerichtete Rechteck, mit der Angabe `align="right"` fließt er links um das rechts ausgerichtete Rechteck.

**Beachten Sie:** Für das, was dieses Element leistet, benutzt man heute CSS Stylesheets. Maßgeblich sind die CSS-Eigenschaften für Außenrand und Abstand.

### 5.12.4 Blinkender Text (Netscape)

Netscape führte in seinen Frühzeiten die Möglichkeit ein, Text blinken zu lassen. Alles was zwischen `<blink>` und `</blink>` steht, blinkt im Netscape-Browser, aber auch nur dort. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1><blink>Warnung!</blink></h1>
</body>
</html>
```

**Beachten Sie:** Blinkenden Text können Sie ganz offiziell mit der CSS-Eigenschaft `text-decoration` erzeugen. Auch die entsprechende Wertzuweisung für blinkenden Text wird jedoch nur von Netscape interpretiert. Für eine Lösung, die mit beiden Browsern funktioniert, müssen Sie mit Dynamischem HTML (Kapitel 21) programmieren.

## 6 Verweise

### 6.1 Verweise definieren und gestalten

#### 6.1.1 Allgemeines zu Verweisen

Verweise sind ein entscheidender Bestandteil jedes Hypertext-Projekts und der »intelligente Mehrwert« des World Wide Web. Mit Hilfe von Verweisen strukturieren Sie Ihr Projekt. Wenn Sie beispielsweise eine HTML-Datei mit einer Projekt-Einstiegsseite und verschiedene HTML-Dateien für Themenseiten planen, brauchen Sie in der Einstiegsseite Verweise zu allen Themenseiten und in jeder Themenseite einen Rücksprungverweis zur Einstiegsseite. Erst dadurch wird aus der losen Dateisammlung ein zusammenhängendes Web-Projekt, eine Homepage. In komplexeren Projekten wie dem vorliegenden Dokument finden Sie auch Verweise mit bestimmter Bedeutung – Verweise zu übergeordneten oder untergeordneten Ebenen, Verweise zum Blättern, Verweise zum Wechseln zwischen Beschreibungen und Anzeigebeispielen, Querverweise, Verweise zu anderen Web-Adressen usw.

Alle Verweise in HTML haben den gleichen Aufbau. Das vereinfacht die Syntax, hat aber auch Nachteile: Es gibt nämlich keine Verweise für bestimmte Aufgaben. Mittlerweile gibt es zwar auch Attribute für solche Zwecke, aber in der Praxis ist es Ihnen überlassen, wie Sie zum Beispiel Verweise so gestalten, dass dem Anwender klar wird, ob es sich um einen Rückverweis zur Einstiegsseite, um einen Querverweis zu einem assoziativ verwandten Thema oder etwa um einen Verweis zu einer ganz anderen Web-Adresse handelt.

- Wichtig ist in allen Fällen ein aussagekräftiger Verweistext.  
**Schlecht** ist: zurück (wohin eigentlich?)  
**Besser** ist: zurück zur Einstiegsseite (ah ja!)
- Verweise lenken die Aufmerksamkeit des Anwenders sofort auf sich. Wenn Sie Verweise mitten im Text notieren, sollten Sie als Verweistext inhaltlich beschreibende Wörter anbieten, keine Wörter ohne Inhalt.  
**Schlecht** ist: Für weitere Information klicken Sie hier (worum geht es »hier« eigentlich?)  
**Gut** ist: Weitere Information können Sie ebenfalls aufrufen (ah ja!)
- Bei komplexeren Projekten sollten Sie sich überlegen, wie Sie dem Anwender die Vielzahl der Verweise mit unterschiedlicher Bedeutung intuitiv zugänglich machen. Ein sinnvoller Weg ist, den Verweisen kleine Symbolgrafiken voranzustellen, so wie es in der HTML-Version dieses Buches auf der Buch-CD auch der Fall ist (siehe auch *Grafiken einbinden*, Kapitel 7). Durch aussagekräftige Symbole können Sie dem Anwender sofort signalisieren, um welche Art von Verweis es sich handelt. Die Symbolgrafiken sollten jedoch in der Höhe nicht größer sein als normaler Fließtext – normalerweise etwa 10-12 Pixel. Auch Grafiken als Verweise (Kapitel 7.3) tragen häufig zu einer besseren Verweiskennzeichnung bei.

- Zu wenige Verweise erschweren das Navigieren in einem Projekt und zeugen von wenig Kreativität beim Anbieter des Projekts. Zu viele Verweise verwirren den Anwender und können ein »Lost-in-Hyperspace-Gefühl« erzeugen – auch das wirft kein gutes Licht auf den Projektanbieter. Setzen Sie deshalb alle Verweise, die zum bequemen Navigieren nötig sind, und denken Sie auch an Querverweise sowie an sinnvolle Verweise zu anderen WWW-Adressen. Übertreiben Sie die Querverweisteknik aber nicht.

### 6.1.2 Beispiele für Verweise

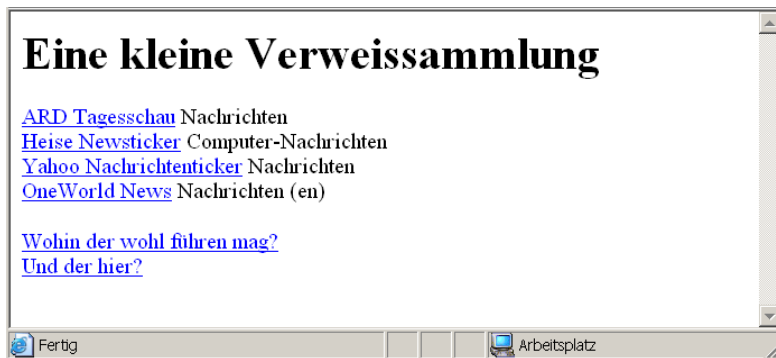
Alle Verweise in HTML haben einen einheitlichen Aufbau, egal ob sie zu einem Verweisziel in der gleichen Datei, zu einer anderen Datei im eigenen Projekt, zu einer beliebigen WWW-Adresse oder zu einer beliebigen Datei eines anderen Dateityps im Internet oder lokal auf dem eigenen Rechner führen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Eine kleine Verweissammlung</h1>

<p>
<a href="http://www.tagesschau.de/">
  ARD Tagesschau
</a>
Nachrichten<br>
<a href="http://www.heise.de/newsticker/">
  Heise Newsticker
</a>
Computer-Nachrichten<br>
<a href="http://de.news.yahoo.com/">
  Yahoo Nachrichtenticker
</a>
Nachrichten<br>
<a href="http://www.oneworld.org/news/today/">
  OneWorld News
</a> Nachrichten (en)
</p>
<p>
<a href="/">Wohin der wohl f&uuml;hren mag?</a><br>
<a href=".">Und der hier?</a>
</p>

</body>
</html>
```



6.1: Verweise führen den Besucher einer Web-Seite zu anderen »Orten« im WWW.

Für Verweise in HTML gibt es das `a`-Element (*a* = *anchor* = *Anker*). Damit jedoch ein Verweis aus diesem Element wird, ist das Attribut `href=` erforderlich (*href* = *hyper reference* = *Hyper(text)-Referenz*). Als Wert an das `href`-Attribut weisen Sie das gewünschte Verweisziel zu. Als Inhalt des `a`-Elements, also zwischen `<a>` und `</a>`, notieren Sie den Text, der dem Anwender als Verweis angeboten wird (bei den meisten Web-Browsern andersfarbig, meist unterstrichen).

Im obigen Beispiel führen die ersten vier Verweise zu anderen Web-Angeboten. Die letzten beiden Verweise führen zu »lokalen« Zielen. Um die Syntax möglicher Verweisziele zu verstehen, müssen Sie sich mit dem Referenzieren in HTML (Kapitel 3.7) vertraut machen.

Beachten Sie: Als Inhalt des `a`-Elements, also bei Verweisen der Verweistext, ist nicht nur reiner Text erlaubt. Sie können im Verweistext auch andere Inline-Elemente notieren. Unter anderem können Sie anstelle von Text auch eine Grafik referenzieren und auf diese Weise Grafiken als Verweise fungieren lassen, was in der Praxis des Web-Designs recht häufig vorkommt.

### 6.1.3 Verweise optisch gestalten mit CSS

Wenn Sie nichts anderes angeben, erhalten Verweise bei der Darstellung die im Browser voreingestellten Farben, z.B. blau für Verweise zu noch nicht besuchten Seiten, und violett für Verweise zu bereits besuchten Seiten. Um dateiweit eigene Verweisfarben zu bestimmen, können Sie die Attribute verwenden, die im Abschnitt Farben für Text und Verweise (Kapitel 4.2.1) beschrieben werden. Darüber hinaus gibt es aber die Möglichkeit, Verweise mit Hilfe von CSS individueller zu gestalten. Unter anderem ist es auch möglich, die meist voreingestellte Unterstreichung des Verweistextes zu unterdrücken. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<style type="text/css">
<!--
a:link { text-decoration:none; font-weight:bold; color:#E00000; }
a:visited { text-decoration:none; font-weight:bold; color:#800000; }
a:hover { text-decoration:none; font-weight:bold; background-color:#FFFF00; }
```

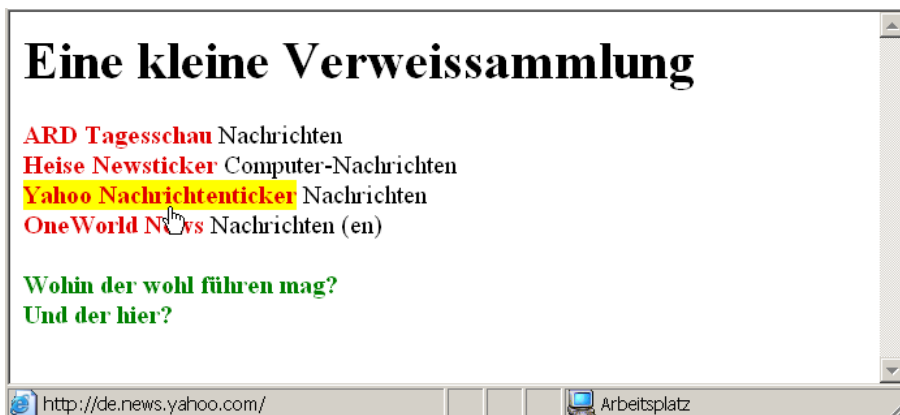
```

a:active { text-decoration:none; font-weight:bold; background-color:#CCFFFF; }
-->
</style>
<title>Text des Titels</title>
</head>
<body>

<h1>Eine kleine Verweissammlung</h1>

<p>
<a href="http://www.tagesschau.de/">
  ARD Tagesschau
</a> Nachrichten<br>
<a href="http://www.heise.de/newsticker/">
  Heise Newsticker
</a> Computer-Nachrichten<br>
<a href="http://de.news.yahoo.com/">
  Yahoo Nachrichtenticker
</a> Nachrichten<br>
<a href="http://www.oneworld.org/news/today/">
  OneWorld News
</a> Nachrichten (en)
</p>
<p>
<a href="/" style="color:green">
  Wohin der wohl f&uuml;hren mag?
</a><br>
<a href="." style="color:green">Und der hier?</a>
</p>
</body>
</html>

```



6.2: Die Links aus dem vorhergehenden Beispiel, diesmal mal mit Hilfe von CSS formatiert.

Das Beispiel arbeitet konsequent mit CSS Stylesheets. Dazu ist im Dateikopf der HTML-Datei ein eigener zentraler Bereich zum Definieren von CSS-Formaten notiert (`<style>...</style>`). Näheres dazu finden Sie im CSS-Kapitel im Abschnitt *Zentrale Formate definieren* (Kapitel 15.3). Der vielfache Wunsch von Web-Designern, die Unterstreichung des Verweistextes zu unterdrücken, wird durch die CSS-Angabe `text-decoration:none` erzielt (`text-decoration:underline` würde übrigens die Voreinstellung, also unterstrichenen Verweistext bewirken). Weitere im Beispiel verwendete Formatdefinitionen sind `font-weight:bold`; (fette Schrift) sowie Angaben zur Farbe (`color:`) und zur Hintergrundfarbe (`background-color:`). Die entsprechenden CSS-Eigenschaften werden in den Abschnitten über Schriftformatierung und Hintergrundfarben und -bilder beschrieben.

Was im obigen Beispiel auffällt, ist, dass für insgesamt vier »Verweistypen« CSS-Formate definiert werden. Mit `a:link` sind Verweise zu noch nicht besuchten Seiten gemeint, mit `a:visited` Verweise zu bereits besuchten Seiten, mit `a:hover` Verweise, während der Anwender mit der Maus über den Verweistext fährt, und mit `a:active` Verweise, wenn sie angeklickt werden oder sind. Weitere Einzelheiten dazu werden im Abschnitt *Pseudoformate für Verweise* (Kapitel 15.3.8) beschrieben.

Die beiden letzten Verweise im obigen Beispiel werden individuell formatiert. Sie übernehmen die zentral definierten CSS-Formate, erhalten aber zusätzlich eine eigene Verweistextfarbe zugewiesen. Dazu wird im einleitenden `<a>`-Tag das Attribut `style=` notiert.

#### 6.1.4 Zielfenster für Verweise bestimmen

Per Voreinstellung werden alle Verweise im aktuellen Fenster geöffnet – es sei denn, Sie haben in den Kopfdaten der HTML-Datei eine Zielfensterbasis angegeben und dort etwas anderes bestimmt. Bei einzelnen Verweisen können Sie jedenfalls unabhängig davon entscheiden, ob ein Verweisziel im aktuellen Browser-Fenster ausgegeben, oder ob ein neues Fenster dafür geöffnet werden soll. Bei Verweisen zu fremden Web-Angeboten kann es beispielsweise sinnvoll sein, das Verweisziel in einem neuen Fenster zu öffnen. So braucht der Anwender das eigene Web-Angebot nicht verlassen, sondern bekommt das andere Angebot einfach in einem neuen Fenster angezeigt. Die Seite Ihres Angebots mit dem Verweis bleibt in einem Fenster im Hintergrund erhalten.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>SELFHTML aktuell</h1>
<p>
immer wieder neu: <a href="http://selfaktuell.teamone.de/" target="_blank">SELFHTML
aktuell</a>
</p>
```



```
</body>
</html>
```

Mit dem Attribut `target=` im einleitenden `<a>`-Tag können Sie ein Zielfenster für den Verweis festlegen. Wenn Sie den Verweis in einem neuen Fenster öffnen wollen, weisen Sie bei `target` den Wert `_blank` zu (mit dem Unterstrich am Anfang!).

Weitere mögliche Wertzuweisungen an `target=` sind:

- `_self`, um den Verweis im aktuellen Fenster zu öffnen,
- `_parent`, um bei verschachtelten Framesets das aktuelle Frameset zu sprengen,
- `_top`, um bei verschachtelten Framesets alle Framesets zu sprengen,
- sowie ein beliebiger Fenstername, der in einem Frameset definiert ist.

Genauere Einzelheiten zu den Angaben im Zusammenhang mit Frames finden Sie im Abschnitt *Verweise zu anderen Frames und zum Sprengen von Framesets* (Kapitel 9.3).

**Beachten Sie:** Das `target`-Attribut ist zwar nicht als *deprecated* gekennzeichnet, doch um es einzusetzen, müssen Sie die HTML-Variante »Transitional« verwenden. Der Grund ist, dass dieses Attribut vorwiegend für Verweise bei Verwendung von Frames gedacht ist und Frames eine eigene HTML-Variante haben, die von der Einstufung her der Variante »Transitional« entspricht (auf gut Deutsch: nicht der »reinen Lehre« entspricht). Anstelle von `target=` `"_blank"` können Sie auch `target="Halligalli"` oder sonst irgendeinen Unsinn notieren. Wenn es kein Fenster mit diesem Namen gibt, öffnen die Web-Browser das Verweisziel in einem neuen Browser-Fenster.

## 6.2 Projektinterne Verweise

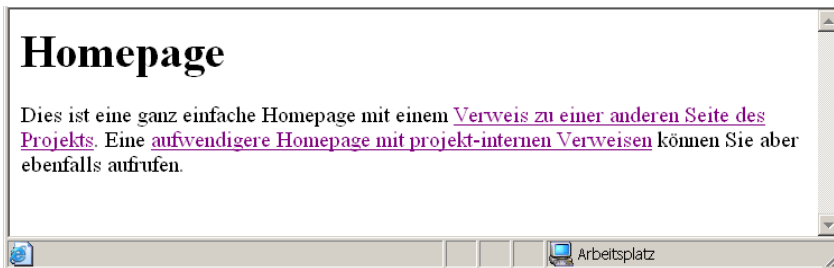
### 6.2.1 Verweise zu Dateien oder Quellen im Projekt

Ein Web-Projekt besteht typischerweise aus mehreren bis vielen Einzelseiten, die miteinander verlinkt sind. Einleitende Bemerkungen dazu finden Sie im Abschnitt *Informationsverteilung und Dateiorganisation* (Kapitel 3.9). Um Verweise auf andere Projektdateien zu definieren, empfiehlt es sich, relative Angaben zum Verweisziel zu machen. Das Projekt bleibt dadurch flexibler, und die Verweise funktionieren auch in anderen Umgebungen (z.B. solange Sie das Projekt lokal auf Ihrem PC erstellen und austesten wollen, oder wenn Sie es mal auf CD-ROM präsentieren möchten).

Beispiel Teil 1 – *index.htm*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
```

```
<h1>Homepage</h1>
<p>Dies ist eine ganz einfache Homepage mit einem <a href="zweiteseite.htm">Verweis
zu einer anderen Seite des Projekts</a>. Eine <a
href="../../tabellen/anzeige/layout.htm">aufwendigere Homepage mit projekt-internen
Verweisen</a> &ouml;nnen Sie aber ebenfalls aufrufen.</p>
</body>
</html>
```



6.3: So sieht die Beispielseite aus: Die Verweise führen auf zwei weitere Web-Seiten.

#### Beispiel Teil 2 – *zweiteseite.htm*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<div style="width:100%; border:thin solid gray; padding:5px">
<a href="index.htm">Homepage</a>
</div>

<h1>Zweite Seite</h1>
<p>Ja, hurra, geschafft!</p>

</body>
</html>
```

Das Beispiel zeigt zwei kleinere HTML-Dateien. Die erste – mit Namen *index.htm* – enthält einen Verweis zur zweiten Datei. Der Dateiname der zweiten Datei lautet *zweiteseite.htm*. Beide Dateien sind im gleichen Verzeichnis abgelegt. Deshalb genügt bei `href=` die Angabe des Dateinamens ohne weitere Zusätze. Im Beispiel der Datei *index.htm* wird aber auch noch gezeigt, wie Verweise zu Dateien in anderen Verzeichnissen definiert werden. Die genaue Syntax solcher relativen Angaben wird im Abschnitt *Mit relativen Pfadangaben relativ zum Basis-URI referenzieren* (Kapitel 3.7.3) beschrieben.

Die zweite Datei im Beispiel (*zweiteseite.htm*) enthält einen typischen »Rückverweis« auf die erste Seite, also auf die »Einstiegsseite«. Ein Rückverweis ist kein spezieller Verweis, sondern einfach wieder ein Verweis auf die gewünschte Datei. Da beide Dateien im gleichen Verzeichnis liegen, genügt auch in diesem Fall wieder die Angabe des Dateinamens, also `href="index.htm"`.

**Beachten Sie:** Der »Rückverweis« in der zweiten Datei wird im obigen Beispiel mit Hilfe eines allgemeinen Bereichs, dem ein paar CSS-Formate zugewiesen sind, optisch etwas aufgewertet. Er steht am Anfang des sichtbaren Dateikörpers und durch den Rahmen, den er dank CSS erhält, wirkt der Bereich wie eine Navigationsleiste. Es ist sehr empfehlenswert, solche typischen Navigationsverweise immer an der gleichen Stelle einer Seite zu notieren und optisch so zu gestalten, dass der Navigationszweck ersichtlich ist. HTML bietet keine speziellen Verweise oder gestalterischen Möglichkeiten für Navigationsleisten und Navigationsverweise an. Navigationsleisten müssen Sie mit den zur Verfügung stehenden Bordmitteln selbst kreieren. (Genaugenommen ist diese Aussage falsch: HTML bietet sehr wohl so etwas an, nämlich mit der Möglichkeit, logische Beziehungen wie in Kapitel 4.1.12 beschrieben zu definieren. Leider wird dies jedoch von den Browsern bis heute nicht unterstützt.)

## 6.2.2 Anker definieren und Verweise zu Ankern

Sie können innerhalb einer HTML-Datei Anker definieren. Dann können Sie Verweise zu solchen Ankern setzen, um einen Sprung genau an die Ankerstelle innerhalb der Datei zu veranlassen. Der Verweis kann in der gleichen Datei stehen. Dann wird einfach ein Sprung innerhalb der angezeigten Seite ausgeführt. Der Verweis kann aber auch in einer anderen Datei stehen. Dann wird die Zieldatei geladen, und der Browser springt, sobald er die Stelle mit dem Anker geladen hat, an die entsprechende Stelle innerhalb der Datei. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1><a name="anfang">Lange Seite</a></h1>

<p><a href="#kap02">Kapitel 2</a></p>
<hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr>
<!-- usw. lauter Trennlinien, um Raum zu schaffen -->

<h2><a name="kap02">Kapitel 2</a></h2>

<p><a href="#anfang">Seitenanfang</a> oder
<a href="../projektintern.htm#anker">
    Anker definieren und Verweise zu Ankern
</a></p>

<hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr>
<!-- usw. lauter Trennlinien, um Raum zu schaffen -->

</body>
</html>
```

Ein Anker wird genau wie ein Verweis mit Hilfe des a-Elements erzeugt. Der Unterschied besteht darin, dass kein Attribut href= notiert wird, sondern stattdessen ein Attribut name=.

Ein kompletter Anker sieht also so aus:

```
<a name="Ankername">...</a>
```

Den Ankernamen können Sie frei vergeben. Vergeben Sie keine zu langen Namen. Namen dürfen keine Leerzeichen und keine deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich ( ), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Was Sie zwischen `<a name="Ankername">` und `</a>` als Inhalt notieren, ist das Sprungziel für Verweise, die zu diesem Anker führen. Es ist durchaus erlaubt, einen leeren Anker zu notieren, also `<a name="Ankername"></a>`. Einige ältere Browser führen Verweise zu leeren Ankern jedoch nicht aus, weshalb es besser ist, den Anker immer um einen konkreten Inhalt zu setzen. Beachten Sie dabei aber, dass das `a`-Element selber ein Inline-Element ist und keine Block-Elemente als Inhalt haben darf. Wenn Sie also beispielsweise eine Überschrift als Anker definieren wollen, was ja durchaus typisch ist, dann notieren Sie die Elementverschachtelung immer in dieser Form:

```
<h2><a name="Ankername">Text der Überschrift</a></h2>
```

Um innerhalb einer Datei einen Verweis zu einem in der Datei vorhandenen Anker zu notieren, gilt folgendes Schema:

```
<a href="#Ankername">Verweistext</a>
```

Das Verweisziel beginnt also mit einem Gatterzeichen `#`, unmittelbar gefolgt vom Ankernamen. Wenn der Verweis zu einem Anker in einer anderen Datei führen soll, wird zuerst die Datei adressiert. Hinter dem Dateinamen folgt das Gatterzeichen `#` und dahinter der Ankername.

**Beachten Sie:** Obwohl HTML nicht zwischen Groß- und Kleinschreibung unterscheidet, ist es in jedem Fall empfehlenswert, Ankernamen bei Ankern und Verweisen dorthin genau gleich zu schreiben. Einige Browser führen den Verweis nämlich nicht aus, wenn der Ankername beim Anker und beim Verweis dorthin unterschiedliche Groß-/Kleinschreibung verwendet. Wenn Sie **XHTML-konform** arbeiten wollen, müssen Sie in jedem Fall auf einheitliche Groß-/Kleinschreibung achten, da XHTML im Gegensatz zu HTML Groß-/Kleinschreibung streng unterscheidet. Bei Verweisen innerhalb einer Datei erzeugt der Web-Browser, wenn die Datei in einer http-Umgebung angezeigt wird, keinen neuen Server-Zugriff, sofern er die Datei so vollständig in den Arbeitsspeicher geladen hat, dass er den Sprung ausführen kann.

Einige Browser, z.B. der Internet Explorer, kennen auch »intern reservierte« Ankernamen wie `top`. Wenn Sie also einen Verweis `<a href="#top">...</a>` notieren und kein Anker dieses Namens in der Datei existiert, springt der Browser beim Ausführen des Verweises an den Anfang der Seite. Manche Browser haben Schwierigkeiten, zu Ankern zu springen, die innerhalb einer Tabelle notiert sind.

Es ist durchaus möglich, `a`-Elemente zu notieren, die sowohl ein `href`- als auch ein `name`-Attribut haben! Absichtlich selbstbezügliche Verweise können Sie beispielsweise so erzeugen:

```
<a name="hier" href="#hier">immer sch&ouml;n hierbleiben!</a>
```

## 6.3 Projektexterne Verweise

### 6.3.1 Allgemeines zu projektexternen Verweisen

Aus technischer Sicht stellen projektexterne Verweise kein großes Problem dar. Im Abschnitt *Referenzieren in HTML* (3.7) wird beschrieben, welche Möglichkeiten zur Referenzierung bestehen. Diese Möglichkeiten sind beim `href`-Attribut des `<a>`-Tags erlaubt. Dennoch sollten Sie einiges mehr über projektexterne Verweise wissen:

- Im Normalfall dürfen Sie ungefragt Verweise auf fremde Web-Angebote setzen. Sie brauchen also keine E-Mail an den Anbieter mit Bitte um Genehmigung zu schreiben, wenn Sie auf sein Angebot einen Link setzen wollen. Jeder Anbieter, der mit seinem Web-Projekt online geht, muss sich im Klaren darüber sein, dass er Teil eines weltweiten Hypertext-Systems ist, in dem er nicht allein ist. Wenn er das nicht akzeptieren kann, ist er im Web fehl am Platz und hat das falsche Medium gewählt. Die Grundregeln des Web werden von den Ideen des Web bestimmt, nicht von den hermetischen Zwangsvorstellungen einiger Zuspätkekommener.
- Es gibt jedoch Ausnahmen von der Regel. Wenn Sie beispielsweise selbst ein sehr stark frequentiertes Web-Angebot haben und auf dessen Einstiegsseite einen Verweis auf die Homepage eines kleinen, unbekannten Anbieters setzen, dann sollten Sie ihn vorher fragen. Der Grund: Durch die vielen zu erwartenden Besucher, die über Ihren Verweis auf das fremde Angebot finden, wird dort plötzlich sehr viel **Traffic** (Besucherverkehr und Datenübertragung) erzeugt. Viele Anbieter haben bei ihrem Provider eine Volumenbegrenzung, und wenn diese überschritten wird, entstehen den Anbietern unkalkulierbare Kosten. Auch könnte es sein, dass der fremde Server nicht sehr belastungsfähig ist und durch die vielen plötzlichen Besucher zusammenbricht.
- Eine andere Ausnahme ist, wenn Sie den Verweis in einem negativen Kontext setzen. Wenn Sie also auf einen Anbieter verweisen, nur weil sie ihn auf Ihren eigenen Seiten heftig kritisieren, dann ist es sicherer, sich dort eine Bestätigung einzuholen, dass ein solcher Verweis gebilligt wird. Juristisch sicher ist eine solche Bestätigung aber nur, wenn sie mit Unterschrift in dokumentenechter Form von einer dafür zuständigen Person geleistet wird.
- Eine Sache, gegen die man sich im Sinne der Zivilcourage wehren sollte, ist der Versuch von Anbietern, juristisch gegen so genannte **Deeplinks** (Verweise in die Tiefe eines fremden Angebots) vorzugehen. Weltweiter Hypertext im Web bedeutet, dass auf vorhandene URIs verwiesen werden darf. Schließlich kann sich auch jeder Anwender auf jeden URI ein Lesezeichen (Bookmark, Favoriten) in seinem Browser setzen. Das kann auch eine Datei tief in einem fremden Web-Angebot sein. Es gibt von Anbieterseite aus technische Möglichkeiten, Deeplinks auf eine allgemeinere Seite umzulenken. Wenn er also nicht will, dass Deeplinks auf sein Angebot gesetzt werden, dann soll er von diesen technischen Möglichkeiten Gebrauch machen.

Weitere Informationen und Bemerkungen zu Diskussionen, die Sie zu diesen Themenkomplexen kennen sollten, finden Sie im Abschnitt *Rechtliche Aspekte von Web-Projekten* (Kapitel 25.6).

### 6.3.2 Beispiele für projektexterne Verweise

Die einzige Bedingung, die beim Anwender erfüllt sein muss, damit er projektexterne Verweise ausführen kann, ist eine bestehende Internet-Verbindung. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Besuchen Sie doch mal...</h1>

<p>
<a href="http://www.teamone.de/training/">
  ein paar Kurse &uuml;ber die hier vermittelten Inhalte
</a><br>
<a href="http://www.atomic-eggs.com/selfspezial/guests/advguest.cgi?view">
  das <i>SELFSpezial</i>-G&auml;stebuch
</a><br>
<a href="http://babelfish.altavista.com/">
  den Babelfish
</a><br>
<a href="ftp://ftp.uni-augsburg.de/">
  den FTP-Server der Universit&auml;t Augsburg
</a><br>
<a href="gopher://ftp.std.com/1">
  die ganze Welt auf einem Gopher-Server
</a><br>
<a href="telnet://locis.loc.gov/">
  die &ouml;ffentliche Telnet-Anwendung der Library of Congress
</a><br>
<a href="news:de.comm.infosystems.www.authoring.misc">
  die deutschsprachige Newsgroup f&uuml;r HTML/CSS
</a>
</p>

</body>
</html>

```

Web-Adressen beginnen mit `http://`, manchmal auch mit `https://` (Letzteres sind Server, bei denen die Datenübertragung von und zum Browser verschlüsselt stattfindet, z.B. bei Internet-Banking).

Andere Internet-Protokolle können Sie ebenfalls adressieren, beispielsweise FTP-Adressen mit `ftp://`, Adressen auf Gopher-Servern mit `gopher://` oder Telnet-Adressen mit `telnet://`. Auch Newsgroups im Usenet können Sie adressieren, nämlich mit `news:` (ohne die beiden sonst charakteristischen Schrägstriche).

Bei anderen als `http`-Adressen kommt es darauf an, wie Web-Browser damit umgeht. Die modernen Browser beherrschen meistens FTP und Gopher und stellen entsprechende Adressen in ihrem Anzeigefenster dar. Bei Protokollen, die der Browser nicht unterstützt, versucht er, auf dem Rechner des Anwenders ein Programm auszuführen, das für das entsprechende Internet-Protokoll zuständig ist. Bei Telnet wird beispielsweise ein auf dem Rechner installierter Telnet-

Client auferufen, und bei Verweisen auf Newsgroups ein Newsreader oder das Newsreader-Modul eines Mailprogramms. Bei Newsgroups muss jedoch ein News-Server im Newsreader des Anwenders eingerichtet sein, der die adressierte Newsgroup anbietet. Auch E-Mail-Verweise sind möglich (Kapitel 6.4).

**Beachten Sie:** Viele Adressen bestehen nur aus dem Namen einer »WWW-Domain«, etwa `http://www.teamone.de/`. Trotzdem führt der Verweis auf eine konkrete HTML-Datei. Das liegt daran, dass es bei vielen Web-Servern einen so genannten Default-Dateinamen gibt – meistens `index.htm`, `index.html` oder `welcome.htm` bzw. `welcome.html`. Das Projekt muss natürlich auch eine entsprechende Datei besitzen. Im Verweis braucht die HTML-Datei aber nicht mit angegeben zu werden.

Viele solcher Adressen werden immer wieder ohne abschließenden Schrägstrich angegeben, etwa `http://www.teamone.de`. Es ist jedoch sauberer, wenn Sie noch den Schrägstrich dahinter setzen. Nur so kann der Web-Browser bereits am Verweis erkennen, dass es sich um ein Verzeichnis handelt, in dem eine Default-Datei steht, deren Namen der Web-Server zur Verfügung stellt. Es ist deshalb besser zu notieren: `http://www.teamone.de/`.

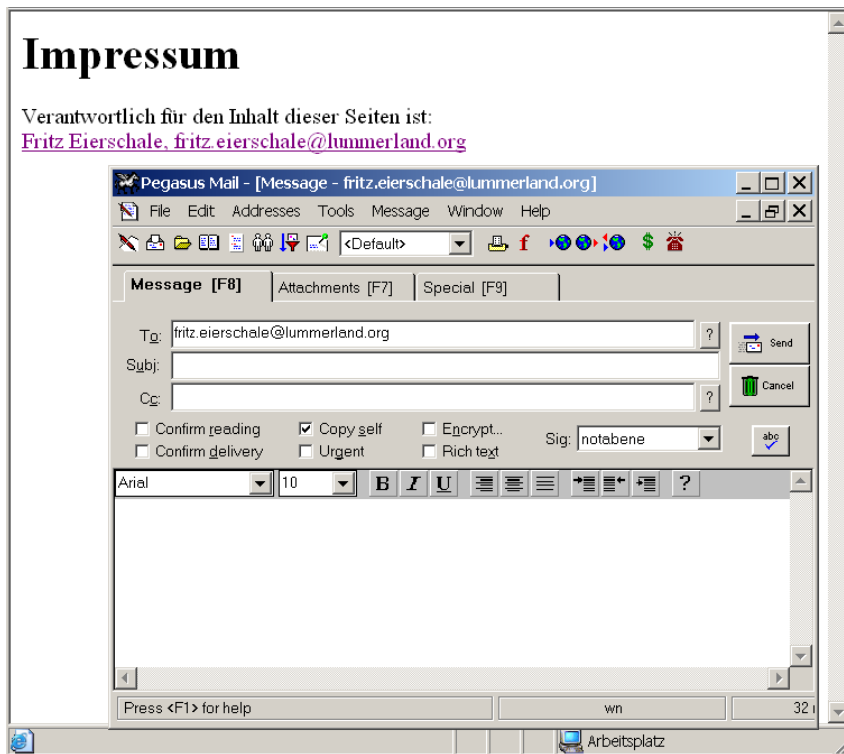
Noch wichtiger ist es, bei Unterverzeichnissen einen abschließenden Schrägstrich zu notieren. Zwar klappt es auch, wenn Sie eine Adresse wie `http://www.teamone.de/training` notieren. Doch dann findet unnötig viel Kommunikation zwischen Browser und Server statt, denn intern fordert der Browser vom Server im Beispiel erst mal eine Datei namens `training`, was den Server zunächst zu einer Fehlermeldung veranlasst, da diese Datei nicht existiert. Erst im zweiten Schritt wird erkannt, dass es sich um den Namen eines Verzeichnisses handelt. Notieren Sie deshalb **immer** Angaben wie `http://www.teamone.de/training/`, also mit abschließendem Schrägstrich.

Es gibt neben den angesprochenen Internet-Protokollen auch noch andere, etwas `wais` oder `irc`. Ferner gibt es das »Un-Protokoll« `file`, über das sich lokale Rechner und Netzwerkadressen absolut adressieren lassen, mit Angaben wie `<a href="file://localhost/">...</a>`. Dies wird von moderneren Browsern jedoch aus Sicherheitsgründen nicht mehr oder nur noch eingeschränkt unterstützt und ist auch nicht besonders sinnvoll, da es heute wirklich kein Problem mehr ist, für lokale Zwecke einen Web-Server lokal einzurichten.

## 6.4 E-Mail-Verweise

### 6.4.1 Verweis zu E-Mail-Adresse definieren

Sie können auf jede beliebige E-Mail-Adresse im Internet-Format einen Verweis setzen. Eine Internet-gerechte E-Mail-Adresse erkennen Sie an dem Zeichen `@` in der Mitte der Adresse. Wenn der Anwender auf den Verweis klickt, kann er eine E-Mail an den betreffenden Empfänger absetzen. Normalerweise benutzen Anbieter von WWW-Seiten diese Möglichkeit, um Besuchern die Möglichkeit zu bieten, eine E-Mail an die eigene Adresse senden, zum Beispiel wegen Feedback zum Angebot. Sie können aber auch Verweise zu anderen E-Mail-Adressen anbieten.



6.4: Ein Klick auf den Mailto-Link öffnet das verknüpfte E-Mail-Programm (hier Pegasus Mail) mit einer neuen Nachricht.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>E-Mail-Verweise definieren</title>
</head>
<body>

<h1>Impressum</h1>

<p>Verantwortlich für den Inhalt dieser Seiten ist:<br>
<a href="mailto:fritz.eierschale@lummerland.org">Fritz Eierschale,
fritz.eierschale@lummerland.org</a>
</p>

</body>
</html>
```

Verweise auf E-Mail-Adressen funktionieren nach dem gleichen Schema wie alle anderen Verweise. Beim href-Attribut des einleitenden <a>-Tags wird das Verweisziel angegeben. Verweis-



ziele vom Typ »E-Mail« beginnen immer mit `mailto:` (ohne `//` dahinter!). Der Verweis ist bei Anwendern nur ausführbar, wenn der Web-Browser das Erstellen und Absenden von E-Mails unterstützt (z.B. Netscape) oder wenn bei solchen Verweisen automatisch ein E-Mail-Programm gestartet wird (dies ist z.B. beim MS Internet Explorer der Fall).

**Beachten Sie:** Es ist sinnvoll, im Verweistext die E-Mail-Adresse noch einmal explizit zu nennen, so wie im obigen Beispiel, damit Anwender, bei denen der E-Mail-Verweis nicht ausführbar ist, auf Wunsch separat eine E-Mail senden können.

### 6.4.2 Optionen bei E-Mail-Verweisen

Die hier beschriebenen Optionen sind nicht im HTML-Standard verankert, widersprechen diesem aber auch nicht unbedingt, solange die Kodierungsregeln für URIs eingehalten werden. Da sie von vielen Browsern interpretiert werden, sollen sie hier erwähnt werden. Sie können:

- E-Mail-Verweise an mehrere Empfänger senden, wahlweise `cc` (sichtbare Kopienempfänger) oder `bcc` (unsichtbare Kopienempfänger),
- anstelle einer einfachen Mail-Adresse ein vollständiges Empfängerschema angeben von der Art: Fritz Eierschale <fritz.eierschale@lummerland.org>
- ein Subject vordefinieren, sodass beim Öffnen des E-Mail-Editors das Betreff-Feld bereits mit einem Text Ihrer Wahl ausgefüllt ist,
- einen Body-Inhalt vordefinieren, sodass beim Öffnen des E-Mail-Editors bereits Text im Nachrichtentext der E-Mail steht, z.B. eine Anrede,
- alle Optionen kombinieren.

Es besteht jedoch keinerlei Garantie, dass alle Möglichkeiten in allen Browsern und allen Kombinationen mit E-Mail-Programmen funktionieren. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Tolle Mails</h1>

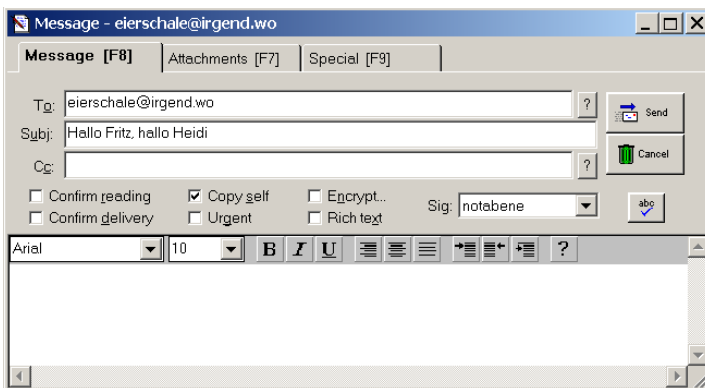
<p>Mail an einen Hauptempf&auml;nger und einen sichtbaren Kopienempf&auml;nger:<br>
<a href="mailto:eierschale@irgend.wo?cc=heidi.bratze@vergiss.es">Mail mit
Kopie</a></p>
<p>Mail an einen Hauptempf&auml;nger und einen unsichtbaren
Kopienempf&auml;nger:<br>
<a href="mailto:eierschale@irgend.wo?bcc=heidi.bratze@vergiss.es">Mail mit
Geheimkopie</a></p>
<p>Mail mit vordefiniertem Subject (Betreff):<br>
<a href="mailto:eierschale@irgend.wo?subject=eine Mail%20von%20deinen%20Web-
Seiten">Mail mit Betreff</a></p>
```

```

<p>Mail mit vordefiniertem Body (Nachrichtentext):<br>
<a
href="mailto:eierschale@irgend.wo?body=Hallo%20Fritz,%0D%0A%0D%0Aich%20wollte%20nur
%20sagen,%20da%DF ">Mail mit Body</a></p>
<p>Mail mit kombinierten Optionen:<br>
<a
href="mailto:eierschale@irgend.wo?cc=heidi.bratze@vergiss.es&subject=Hallo%20Fritz,%20hallo%20Heidi">cc und Subject</a></p>
<p>Mail an zwei Hauptempfänger:<br>
<a href="mailto:eierschale@irgend.wo,%20heidi.bratze@vergiss.es">Mail an
zwei</a></p>
<p>Mail mit vollständigem Adressierungsschema:<br>
<a href="mailto:Fritz%20Eierschale%20<eierschale@irgend.wo>";">Mail mit
Adressierungsschema</a></p>

</body>
</html>

```



6.5: Dieses Beispiel gibt Empfängeradresse und den Betreff der E-Mail vor.

Die Optionen werden in dem Wert notiert, der dem `href`-Attribut zugewiesen wird. Es gibt also keine speziellen HTML-Attribute für diese Optionen, und deshalb ist die Handhabung auch etwas umständlich. Zuerst wird wie üblich der Empfänger notiert. In den obigen Beispielen (mit Ausnahme des letzten) geschieht das jeweils durch `mailto:eierschale@irgend.wo`. Dahinter wird ein Fragezeichen `?` notiert. Das ist in der URI-Syntax das übliche Zeichen, um Parameter an eine aufgerufene Adresse zu übergeben. Hinter dem Fragezeichen folgen die Parameter. Jede Option ist so ein Parameter und besteht aus einem Namen, einem Gleichheitszeichen und einem zugewiesenen Wert. Als Optionsnamen sind erlaubt:

- `cc` (sichtbarer Kopienempfänger),
- `bcc` (unsichtbarer Kopienempfänger),
- `subject` (Betreff) und
- `body` (Nachrichtentext).

Ein Konstrukt wie `cc=heidi.bratze@vergiss.es` ist also ein vollständiger Parameter und bedeutet: »sichtbare Kopie an heidi.bratze@vergiss.es«.

Bei den Wertzuweisungen an die Parameter können Zeichen vorkommen, die nicht zu einer gültigen URI gehören. Damit die URI gültig bleibt (andernfalls wäre das Dokument auch kein gültiges HTML mehr), müssen diverse Zeichen maskiert werden. Die Maskierung besteht darin, ein Prozentzeichen % zu notieren, gefolgt vom hexadezimal ausgedrückten Zeichenwert des gewünschten Zeichens.

Die folgende Tabelle listet Zeichen auf, die maskiert werden müssen, weil sie innerhalb von URIs nicht vorkommen dürfen oder eine bestimmte Bedeutung haben. Links steht das Zeichen, rechts die Zeichenkette, mit der Sie das Zeichen maskieren müssen:

| Zeichen         | Zeichenkette für Maskierung |
|-----------------|-----------------------------|
| [neue Zeile]    | %0A                         |
| [Wagenrücklauf] | %0D                         |
| [Leerzeichen]   | %20                         |
| !               | %21                         |
| #               | %23                         |
| %               | %25                         |
| *               | %2A                         |
| /               | %2F                         |
| ?               | %3F                         |

Ferner müssen Sie alle Zeichen maskieren, die oberhalb des ASCII-Zeichensatzes liegen, also z.B. deutsche Umlaute und scharfes S. Die folgende Tabelle listet die wichtigsten Zeichen und ihre Maskierung auf:

| Zeichen | Zeichenkette für Maskierung | Zeichen | Zeichenkette für Maskierung |
|---------|-----------------------------|---------|-----------------------------|
| Ä       | %C4                         | ä       | %E4                         |
| Ö       | %D6                         | ö       | %F6                         |
| Ü       | %DC                         | ü       | %FC                         |
| ß       | %DF                         |         |                             |

Zur Ermittlung weiterer Zeichen können Sie beispielsweise in der Zeichentabelle des Zeichensatzes ISO-8859-1 den Dezimalwert des gewünschten Zeichens ermitteln, von dieser Dezimalzahl dann mit dem Dezimal/Hexadezimal-Umrechner (auf der Buch-CD in selfHTML) den Hexadezimalwert ermitteln und noch ein Prozentzeichen davor notieren.

Wenn Sie mehrere Optionen kombinieren wollen, müssen Sie die Optionen durch ein kaufmännisches Und (&) voneinander trennen. Nun darf ein solches Zeichen in HTML – auch nicht bei einer Wertzuweisung an ein Attribut – nicht unmaskiert vorkommen. Deshalb sollten Sie es wie in HTML üblich mit `&amp;` maskieren. Leider gibt es einige ältere Browser (z.B. Netscape 3.x), die damit nicht klarkommen.

Die folgende Zeichenkette:

```
cc=heidi.bratze@vergiss.es&amp;subject=Hallo%20Fritz,%20hallo%20Heidi
```

kombiniert im obigen Beispiel also die Optionen für `cc` und `subject`. Um weitere Optionen hinzufügen, notieren Sie ein weiteres `&amp;` und dahinter eine weitere Option, bestehend aus Optionsname, Gleichheitszeichen und zugewiesenem Wert.

Das letzte der obigen Beispiele zeigt, wie Sie ein vollständiges Adressierungsschema gemäß RFC822 als `mailto`-Adresse angeben. Ein solches Schema, wie Sie es vermutlich aus Ihrer E-Mail-Korrespondenz kennen, lautet beispielsweise:

```
Fritz Eierschale <eierschale@irgend.wo>
```

In einem E-Mail-Verweis muss die Zuweisung an das `href`-Attribut jedoch so aussehen:

```
mailto:Fritz%20Eierschale%20&lt;eierschale@irgend.wo&gt;
```

Auch in diesem Fall müssen Sie also alle weiter oben erwähnten Zeichen maskieren. Außerdem müssen Sie die beiden Zeichen `<` und `>` mit ihren HTML-gerechten Maskierungen `&lt;` bzw. `&gt;` umschreiben.

## 6.5 Download-Verweise und Verweise auf beliebige Dateien

### 6.5.1 Download-Verweise

Es gibt keine spezifische Notation in HTML, um Dateien beim Anklicken zum Downloaden anzubieten. Es gibt lediglich Dateitypen, die (fast) jeder Web-Browser so interpretiert, dass er dem Anwender anbietet, die Datei downzuloaden. Das bekannteste Dateiformat dafür ist heute das ZIP-Format (\*.zip). ZIP-Dateien sind Archivdateien, die mehrere andere Dateien enthalten können, sogar ganze Verzeichnisstrukturen. Die enthaltenen Dateien werden außerdem komprimiert. Der Anwender muss die ZIP-Datei nach dem Download mit einem geeigneten Programm entpacken (»Unzip«-Programm). Ein Beispiel:

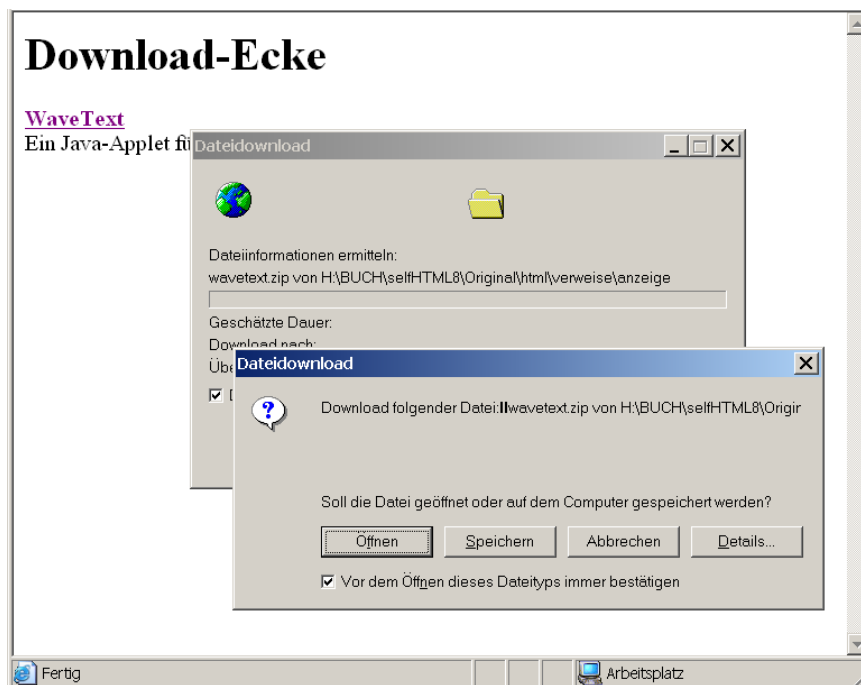
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Download-Ecke</h1>

<p><a href="wavetext.zip"><b>WaveText</b></a><br>
Ein Java-Applet f&uuml;r Textanimationen (wavetext.zip, 2.106 Bytes)</p>

</body>
</html>
```

Verweise auf typische Download-Dateien wie ZIP-Dateien unterscheiden sich nicht von anderen Verweisen. Das Verweisziel ist die Datei, die zum Download angeboten wird. Wenn die Datei zum eigenen Projekt gehört, gelten die gleichen Regeln wie bei projektinternen Verweisen, bei entfernten Dateien die Regeln von projektexternen Verweisen (Kapitel 6.2 und 6.3).



6.6: Nach dem Anklicken eines Download-Links fragt der Internet Explorer 6.0, was mit der Datei passieren soll.

**Beachten Sie:** Wenn Sie Dateien anbieten, die nur für bestimmte Rechnerumgebungen gedacht sind, können Sie natürlich auch Formate verwenden, die speziell für diese Umgebungen gedacht sind. Für DOS/Windows-Umgebungen können das beispielsweise selbstentpackende EXE-Archive sein, oder für Macintosh HQX-Archive. Wenn der Web-Browser mit der Dateierung nichts anfangen kann, sollte er den Anwender im Dialog entscheiden lassen, was er mit der Datei tun möchte – dabei sollte auch die Download-Möglichkeit angeboten werden.

### 6.5.2 Verweise zu beliebigen Dateien

Sie können auf jede beliebige Datei einen Verweis setzen. Es kann sich um Audio-Dateien, Tabellenkalkulationsdateien, CAD-Dateien, Video-Dateien, Grafikdateien, Textverarbeitungsdateien, Programmdateien, Datenbankdateien handeln – was Sie wollen. Sie können auch jede beliebige Datei in Ihr eigenes Web-Projekt mit auf den Internet-Server laden und einen Verweis darauf setzen. Aus Sicht von HTML ist das kein Problem. Das Problem besteht darin, was der Web-Browser des Anwenders mit den Dateien anfangen kann bzw. wie er ihren Inhalt korrekt anzeigen oder abspielen lassen kann. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
```

```

</head>
<body>

<h1>Stuff</h1>

<p><a href="buch.url"><b>buch.url</b></a><br>
Eine Favoritendatei, wie der Internet Explorer sie anlegt</p>
<p><a href="fritz.vcf"><b>fritz.vcf</b></a><br>
Eine Visitenkartendatei (Adressenaustauschformat)</p>
<p><a href="einsundeins.xls"><b>einsundeins.xls</b></a><br>
Eine Excel-Datei</p>

</body>
</html>

```

Verweise auf beliebige Dateien unterscheiden sich nicht von anderen Verweisen. Das Verweisziel ist die gewünschte Datei. Wenn die Datei zum eigenen Projekt gehört, gelten die gleichen Regeln wie bei projektinternen Verweisen, bei entfernten Dateien die Regeln von projektexternen Verweisen.

**Beachten Sie:** Dateitypen wie reine Textdateien (\*.txt) kann der Web-Browser selbst anzeigen. Moderne Web-Browser haben eine Plug-In-Schnittstelle. Wenn der Anwender ein Plug-In zur Darstellung des Dateityps besitzt, kann der Browser die Datei mit Hilfe des Plug-Ins selbst anzeigen bzw. abspielen. Wenn der Anwender ein Programm besitzt, das den Dateityp verarbeiten kann, und dem Web-Browser ist die Verknüpfung zwischen Dateien mit der Endung des Verweisziels und einem Programm bekannt, das solche Dateien verarbeitet, dann kann der Browser das Programm starten. Wenn das Betriebssystem, der Web-Browser und das andere Programm den dynamischen Datenaustausch zwischen Programmen erlauben, kann das Anzeigefenster des Fremdprogramms in das Browser-Fenster eingebettet werden. Es gibt jedoch keine Möglichkeit, solche Dinge in irgendeiner Weise als Web-Autor zu beeinflussen. Wenn der Browser mit dem Dateityp gar nichts anfangen kann, sollte er dem Anwender einen Dialog anbieten, um zu entscheiden, was mit der Datei geschehen soll. Der Anwender sollte die Datei z.B. downloaden können (siehe Bild 6.6).

### 6.5.3 Mime-Type des Verweisziels angeben

Um dem Browser die Aufgabe zu erleichtern, zu erkennen, um welche Art von Datei es sich handelt, können Sie den Mime-Type der Datei angeben, auf die Sie verweisen. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Stuff mit Typ</h1>

<p><a href="einsundeins.xls" type="application/msexcel"><b>einsundeins.xls</b></a><br>

```

```
Eine Excel-Datei</p>
</body>
</html>
```

Mit dem zusätzlichen Attribut `type=` können Sie den Mime-Type der Zielfeile bestimmen, im obigen Beispiel `application/msexcel` für Excel-Dateien.

## 6.6 Typisierte Verweise

### 6.6.1 Logische Beziehung zum Verweisziel angeben

Ein Feature, das Hypertext-Enthusiasten an HTML bislang vermisst haben, ist seit HTML 4.0 Bestandteil der Sprache. Sie können bei einem Verweis angeben, in welcher logischen Beziehung er zu seinem Verweisziel steht. Dazu gibt es eine Reihe vorgegebener Beziehungstypen. Ein Browser könnte beispielsweise beim Überfahren des Verweises mit der Maus ein kleines Icon anzeigen, welches das Verweisziel symbolisiert. Doch weder Netscape noch der Internet Explorer bieten etwas Derartiges bislang an. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<div style="width:100%; background-color:#EEEEEE; padding:5px">
<a href="../../../index.htm" rev="contents">Inhalt</a> |
<a href="../tastatur.htm" rel="next">weiter</a> |
<a href="../dateitypen.htm" rel="prev">zur&uuml;ck</a> |
<a href="../../index.htm" rev="chapter">zum Kapitelanfang</a>
</div>
<h1>Viel Information</h1>
<p>Hoffentlich nicht zu viel...</p>

</body>
</html>
```

Mit dem Attribut `rel=` bestimmen Sie eine logische Vorwärtsbeziehung zum Verweisziel, mit `rev=` eine logische Rückwärtsbeziehung (*rel = relation = Bezug, rev = reverse = Umkehr*). Beide Angaben sind nur in Verbindung mit dem Attribut `href=` sinnvoll – dort geben Sie wie üblich das eigentliche Verweisziel an. Im obigen Beispiel werden diese Attribute auf Verweise einer typischen Navigationsleiste angewendet. Die Navigationsleiste im Beispiel wird mit Hilfe von CSS Stylesheets gestaltet. Folgende logische Bezüge können Sie angeben:

| Schreibweise     | Bedeutung                                                                                                                             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| rel="contents"   | Verweis zum Inhaltsverzeichnis (contents = Inhaltsverzeichnis)                                                                        |
| rev="contents"   | Rückverweis zum Inhaltsverzeichnis                                                                                                    |
| rel="chapter"    | Verweis zum Kapitel (chapter = Kapitel)                                                                                               |
| rev="chapter"    | Rückverweis zum Kapitel                                                                                                               |
| rel="section"    | Verweis zum Abschnitt (section = Abschnitt)                                                                                           |
| rev="section"    | Rückverweis zum Abschnitt                                                                                                             |
| rel="subsection" | Verweis zum Unterabschnitt (subsection = Unterabschnitt)                                                                              |
| rev="subsection" | Rückverweis zum Unterabschnitt                                                                                                        |
| rel="index"      | Verweis zum Stichwortverzeichnis                                                                                                      |
| rev="index"      | Rückverweis zum Stichwortverzeichnis                                                                                                  |
| rel="glossary"   | Verweis zum Glossar                                                                                                                   |
| rev="glossary"   | Rückverweis zum Glossar                                                                                                               |
| rel="appendix"   | Verweis zum Anhang (appendix = Anhang)                                                                                                |
| rev="appendix"   | Rückverweis zum Anhang                                                                                                                |
| rel="copyright"  | Verweis zur Copyright-Angabe                                                                                                          |
| rev="copyright"  | Rückverweis zur Copyright-Angabe                                                                                                      |
| rel="next"       | Verweis zur nächsten Datei in der »Guided tour« (next = nächste Seite)                                                                |
| rev="next"       | Rückverweis zur nächsten Datei                                                                                                        |
| rel="prev"       | Verweis zur vorherigen Datei in der »Guided tour« (prev = previous = vorherige Seite)                                                 |
| rev="prev"       | Rückverweis zur vorherigen Datei                                                                                                      |
| rel="start"      | Verweis zur ersten Datei in der »Guided tour« (start = erste Seite)                                                                   |
| rev="start"      | Rückverweis zur ersten Datei                                                                                                          |
| rel="help"       | Verweis zum Hilfekontext (help = Hilfe)                                                                                               |
| rev="help"       | Rückverweis zum Hilfekontext                                                                                                          |
| rel="bookmark"   | Verweis zu einem allgemeinen Orientierungspunkt (bookmark = Lesezeichen)                                                              |
| rev="bookmark"   | Rückverweis zu einem allgemeinen Orientierungspunkt                                                                                   |
| rel="alternate"  | Verweis zu einer Datei mit dem gleichen Inhalt wie der aktuellen, jedoch in einer anderen Dokumentversion (alternate = alternierend). |
| rev="alternate"  | Rückverweis zu einer Datei mit dem gleichen Inhalt wie der aktuellen, jedoch in einer anderen Dokumentversion                         |

### 6.6.2 Sprache und Zeichensatz des Verweisziels angeben

Sie können auch angeben, in welcher Landessprache das Verweisziel geschrieben ist und welchen Zeichensatz die Zieldatei benutzt. Die Angabe zur Landessprache kann für den Anwender ein nützlicher Hinweis sein. Die Angabe zum Zeichensatz ist interessant, um den Browser schon mal innerlich auf die Internationalisierungshürden vorzubereiten, die ihn beim Verweisziel erwarten. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
```



```

</head>
<body>

<h1>Aus der weiten Welt</h1>

<p><a href="http://www.asharqalawsat.com/" hreflang="ar" charset="ISO-8859-6">Asharq Al-Awsat</a><br>
Saudi-Arabische Zeitung in arabischer Sprache</p>

</body>
</html>

```

Mit dem Attribut `hreflang=` können Sie die Sprache angeben (*hreflang* = *hyper reference language* = *Verweiszielsprache*), und zwar in Form eines Sprachenkürzels (Anhang A.3). Im obigen Beispiel wird `ar` für »arabisch« angegeben.

Mit `charset=` können Sie den Zeichensatz angeben, der auf der Zieladresse verwendet wird (*charset* = *Zeichensatz*). Erlaubt sind Zeichensatzangaben, wie sie auf der Web-Adresse <http://www.iana.org/assignments/character-sets> angegeben sind. Mehr zu diesem Thema finden Sie innerhalb dieser Dokumentation bei Zeichensätzen (ISO-8859-Familie und andere) im Anhang. Im obigen Beispiel wird ISO-8859-6 angegeben – das ist der ISO-Zeichensatz für arabische Schrift.

**Beachten Sie:** Ob und wie ein Browser solche Angaben für den Anwender sichtbar macht, ist nicht festgelegt. Ein Browser könnte beispielsweise beim Überfahren des Verweises mit der Maus ein kleines Flaggen-Icon anzeigen, das die Landessprache symbolisiert. Bei der Angabe zum Zeichensatz könnte der Browser beim Anklicken des Verweises beispielsweise eine Warnung ausgeben, dass der Zeichensatz auf dem Anwender nicht verfügbar ist, und ihn in einem Dialog fragen, ob er den Verweis wirklich ausführen will. Weder Netscape noch MS Internet Explorer bieten etwas Derartiges bislang an.

## 6.7 Tabulator-Reihenfolge und Tastaturkürzel

### 6.7.1 Tabulator-Reihenfolge für Verweise

Modernere Browser erlauben es, mit Hilfe der Tabulator-Taste an der Tastatur nacheinander die Verweise einer HTML-Datei anzuspringen (mit der Return-Taste ist der Verweis dann ausführbar). Normalerweise werden die Verweise dabei in der Reihenfolge angesprungen, in der sie in der Datei definiert sind. Sie können in HTML jedoch eine andere Reihenfolge festlegen, zum Beispiel, um beim Anspringen von Verweisen zuerst Projektverweise und erst dann Verweise zu anderen WWW-Seiten anzubieten. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

```

```

<h1>Steppen Sie mal mit [Tab] durch</h1>

<p>
<a href="http://german.imdb.com/" tabindex="4">
  Movie Database
</a><br>
<a href="http://www.chart-lyrics.com/" tabindex="2">
  Chart Lyrics
</a><br>
<a href="http://www.meinestadt.de/" tabindex="1">
  Meine Stadt
</a><br>
<a href="http://www.paperball.de/" tabindex="3">
  Paperball
</a>
</p>

</body>
</html>

```

Mit dem Attribut `tabindex=` im einleitenden Verweis-Tag können Sie Angaben zur Tabulator-reihenfolge machen. Notieren Sie die Angabe in allen Verweisen und vergeben Sie bei jeder Angabe eine Zahl. Beim Anspringen der Verweise mit der Tabulator-Taste wird zuerst der Verweis mit der niedrigsten Tabindex-Nummer angesprungen, dann der mit der zweitniedrigsten usw. und als Letztes der Verweis mit der höchsten Tabindex-Nummer. Im obigen Beispiel wird also zuerst der dritte Verweis angesprungen, dann der zweite, dann der vierte und zuletzt der erste. Es sind Zahlen zwischen 0 und 32767 erlaubt.

**Beachten Sie:** Die Tabindizes beziehen sich stets auf die gesamte angezeigte Datei. Dabei werden auch Formulare, Verweisbereiche in verweissensitiven Grafiken und Objekte mit einbezogen. Wenn Sie außer normalen Verweisen auch solche Elemente in Ihrer Datei haben, sollten Sie die Tabulator-Reihenfolge für alle Elemente gemeinsam festlegen. Siehe auch *Tabulator-Reihenfolge bei Formularen* in Kapitel 10.9.1.

### 6.7.2 Tastaturkürzel für Verweise

Sie können dem Anwender anbieten, mit einem Tastendruck zu einem bestimmten Verweis zu springen. Der Verweis wird damit angesprungen – ausgeführt werden kann er anschließend durch Betätigen der Return-Taste. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

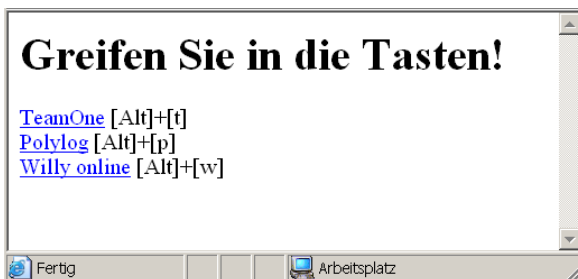
<h1>Greifen Sie in die Tasten!</h1>

```

```

<p>
<a href="http://www.teamone.de/" accesskey="t">
  TeamOne
</a> [Alt]+[t]<br>
<a href="http://www.polylog.org/" accesskey="p">
  Polylog
</a> [Alt]+[p]<br>
<a href="http://www.willy-online.de/" accesskey="w">
  Willy online
</a> [Alt]+[w]
</p>
</body>
</html>

```



6.7: Wenn Sie Links Tastaturkürzel zuweisen, sollten Sie diese deutlich auf der Web-Seite notieren.

Mit dem Attribut `accesskey=` im einleitenden Verweis-Tag können Sie ein Zeichen auf der Tastatur bestimmen, das der Anwender drücken kann, um den Verweis direkt anzuspringen (*accesskey = Zugriffstaste*). Bei den meisten Browsern ist es so geregelt, dass Sie bei `accesskey=` einen Buchstaben angeben können, der dann mit [Alt] und der Taste für den Buchstaben direkt anwählbar ist.

**Beachten Sie:** Benutzen Sie Tasten, die auf jeder Tastatur zu finden sind. Funktionstasten etwa, wie es sie auf MF-II-Tastaturen für MS Windows gibt, können Sie mit dieser Angabe gar nicht ansprechen. Weisen Sie den Anwender im Text auf die Möglichkeit des Tastaturzugriffs hin. Benutzen Sie dann auch einheitliche Tastaturkürzel, beispielsweise auf jeder Seite ein `h` für den Rückverweis zur Homepage.

## 7 Grafiken

### 7.1 Grafiken einbinden

#### 7.1.1 Allgemeines zu Grafiken

Um Grafiken in Ihre Web-Seiten einzubinden, referenzieren Sie die Grafikdateien an gewünschten Stellen im HTML-Quelltext. Geeignete Dateiformate für webgerechte Grafiken sind vor allem GIF und JPEG, allmählich aber auch PNG (Einzelheiten zu diesen Formaten siehe *Grafikformate für Web-Seiten*, Kapitel 24.1).

Wenn Sie HTML-Dateien fürs Web erstellen, sollten Sie darauf achten, dass die darin referenzierten Grafiken nicht zu groß sind, denn aufwändige Grafiken verursachen lange Ladezeiten und Missmut beim Anwender. Reduzieren Sie in Ihren Grafiken gegebenenfalls die Anzahl der Farben, verringern Sie die Bildgröße und stopfen Sie nicht zu viele Grafikreferenzen in eine einzige HTML-Datei. In jedem Fall sollten Sie Breite und Höhe des Bildes mit angeben.

Grafiken sind aus HTML-Sicht Inline-Elemente. In der »Strict«-Variante von HTML müssen solche Elemente innerhalb von Block-Elementen vorkommen, etwa in einem Textabsatz oder einem allgemeinen Bereich oder auch einer Tabellenzelle. Unter grafischen Benutzeroberflächen wie MS-Windows, Macintosh, KDE und anderen können Web-Browser die referenzierten Grafiken direkt im Anzeigefenster anzeigen. Web-Browser für textorientierte Oberflächen (z.B. DOS-Textmodus, Unix-Textmodus) müssen ein separates Grafik-Betrachtungsprogramm aufrufen, um eine Grafik anzuzeigen.

Wenn Sie wichtige HTML-Dateien Ihres Web-Projekts, etwa die Einstiegsseite, vollständig grafisch gestalten, also fast keinen Text, dafür nur Grafikreferenzen verwenden, erschweren Sie automatischen Suchprogrammen, die von vielen Suchdiensten im Web eingesetzt werden, die Identifizierung des Inhalts Ihres Projekts. Im Zweifelsfall müssen Sie abwägen, was Ihnen wichtiger ist: das grafische Erscheinungsbild oder die inhaltliche Identifizierbarkeit. Falls Sie wichtige HTML-Dateien Ihres Web-Projekts vorwiegend durch Grafikreferenzen gestalten, sollten Sie zumindest die Möglichkeit der Meta-Angaben zum Inhalt (Kapitel 4.1.2) nutzen.

#### 7.1.2 Grafikreferenz definieren

An einer gewünschten Stelle können Sie eine Grafik referenzieren. Referenzieren bedeutet, dass Sie den Dateinamen in einem Tag eintragen, damit der Browser weiß, welche Bilddatei er anzeigen soll. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```

```

<title>Text des Titels</title>
</head>
<body>

<h1>Tanzmaus</h1>

<p></p>

</body>
</html>

```



7.1: Das obige Beispiel referenziert eine Bilddatei und bindet sie damit in eine Web-Seite ein.

Für Grafikreferenzen gibt es in HTML das `<img>`-Tag (*img* = *image* = *Bild*, *src* = *source* = *Quelle*). Es handelt sich um ein Standalone-Tag ohne Elementinhalt und ohne End-Tag. Mit Hilfe von Attributen bestimmen Sie nähere Einzelheiten der Grafikreferenz. Zwei Attribute sind Pflicht und müssen immer angegeben werden:

- Mit dem Attribut `src=` bestimmen Sie die gewünschte Grafikdatei. Dabei gelten die Regeln zum Referenzieren in HTML (Kapitel 3.7). Wenn sich die Grafikdatei also beispielsweise im gleichen Verzeichnis wie die HTML-Datei befindet, in der die Grafikreferenz steht, dann genügt einfach die Angabe des Dateinamens der Grafik – so wie im obigen Beispiel. Das Referenzieren mit relativen oder absoluten Pfadangaben ist jedoch ebenso möglich, z.B.:

```







```

Auch das Referenzieren von einem vollständigen URI ist möglich, z.B.:

```



```

- Beim Attribut `alt=` geben Sie einen Alternativtext an für den Fall, dass die Grafik nicht angezeigt werden kann. Es gibt viele Gründe, warum eine Grafik nicht angezeigt werden kann, z.B. weil der Anwender das Anzeigen von Grafiken in seinem Browser deaktiviert hat oder weil der Anwender das Laden der Seite abgebrochen hat, bevor die Grafik übertragen wurde. Der Alternativtext kann ein einzelnes Wort sein, aber auch eine Kurzbeschreibung, etwa so:

```

```

Das `alt`-Attribut ist nach HTML-Standard Pflicht. Wenn Sie aus irgendwelchen Gründen keinen Alternativtext notieren wollen, z.B. weil es sich bei der Grafik um einen so genannten »blinden Pixel« handelt, dann notieren Sie `alt=""` – also eine leere Zeichenkette als Zuweisung.

**Beachten Sie:** Sie können also ohne weiteres Grafiken von fremden Adressen in Ihre Seiten einbinden. Fragen Sie in diesem Fall jedoch unbedingt den Anbieter der Grafik um Erlaubnis, denn das Einbinden von Grafiken ist – im Gegensatz zu Verweisen auf fremde Inhalte – ein Zueigenmachen von fremden Inhalten. Auch besteht keine Garantie, dass die Grafikreferenz jederzeit bis in alle Zukunft funktioniert, denn wenn der Fremdanbieter die Grafik bei sich entfernt oder umbenennt, wird auf Ihrer Seite nur noch ein so genanntes broken image angezeigt.



7.2: Kann der Browser die Bilddatei nicht finden oder anzeigen, so erscheint statt dessen ein Fehlerbild (hier im Internet Explorer).

Achten Sie bei der Wertzuweisung an das `src`-Attribut unbedingt auf Groß-/Kleinschreibung von Dateinamen und Verzeichnisnamen. Die meisten Server-Rechner laufen mit Betriebssystemen, bei denen streng zwischen Groß- und Kleinschreibung unterschieden wird. Am einfachsten und sichersten ist es, wenn Sie alle Datei- und Verzeichnisnamen kleinschreiben – sowohl beim Vergeben der Namen als auch beim Referenzieren in der HTML-Datei.

Netscape interpretiert ein proprietäres Attribut `lowsrc=`, das Sie zusätzlich zum `src`-Attribut notieren können und bei dem Sie eine Grafikdatei angeben, die weniger Speicher benötigt als die bei `src=` referenzierte Grafik. Die `lowsrc`-Grafik lädt der Browser dann zuerst und kann sie schon anzeigen, während die andere Grafik noch geladen wird. Dieses Attribut erzeugt jedoch ungültiges HTML und hat sich in der Praxis auch als überflüssig erwiesen.

Wenn Sie **XHTML-konform** arbeiten, müssen Sie das `img`-Element als inhaltsleer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<img... />`. Weitere Informationen dazu im Kapitel 14.1.11 über *XHTML und HTML*.

### 7.1.3 Breite und Höhe von Grafiken

Wenn Sie Grafiken in HTML-Dateien einbinden, die Sie im Web anbieten wollen, sollten Sie stets die Breite und Höhe der Grafik mit angeben. Dadurch entnimmt der Web-Browser bereits der HTML-Datei, wie groß die Grafik ist, und muss nicht warten, bis er die entsprechende Header-Information der Grafikdatei ausgelesen hat. So kann er die gesamte Web-Seite bereits am Bildschirm aufbauen und bei noch nicht eingelesenen Grafiken erst mal eine entsprechend große Freifläche anzeigen. Wenn Sie Breite und Höhe nicht angeben, wartet der Browser dagegen mit der Anzeige der Web-Seite, bis er alle nötigen Größenangaben aus eingebundenen Grafikdateien eingelesen hat, oder er muss den Bildschirmaufbau korrigieren, was nicht sehr schön aussieht. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Tanzb&auml;r</h1>

<p></p>

</body>
</html>
```

Mit dem Attribut `width=` [Pixel] geben Sie die Breite der Grafik an, und mit `height=` [Pixel] die Höhe (*width* = Breite, *height* = Höhe). Um die genaue Breite und Höhe einer Grafik zu ermitteln, brauchen Sie entweder ein Grafikprogramm, das diese Werte anzeigt, oder einen HTML-Editor, der beim Einbinden einer Grafik im Dialog auch gleich den Dateikopf der Grafik ausliest und die entsprechenden Angaben in das `<img>`-Tag automatisch einfügt.

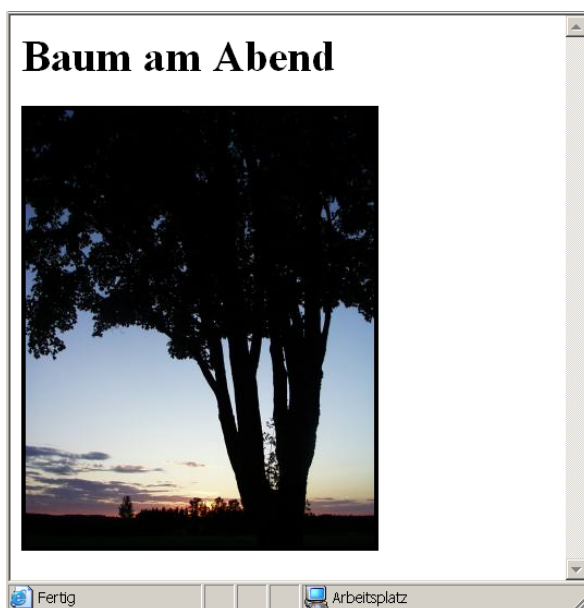
**Beachten Sie:** Sie können auch Angaben zu Breite und Höhe einer Grafik machen, die absichtlich von den tatsächlichen Abmessungen der Grafik abweichen. Dadurch können Sie interessante Verzerrungseffekte erreichen, wie man sie von Cartoons kennt.

Neben Pixelangaben sind bei `width=` und `height=` auch Prozentangaben erlaubt. Solche Angaben beziehen sich auf den zur Verfügung stehenden Raum. Wenn sich die Grafik also beispielsweise in einem normalen Textabsatz befindet, bewirkt die Angabe `width="100%"`, dass die Grafik über die gesamte Breite des Anzeigefensters (abzüglich der Seitenränder der angezeigten Web-Seite) dargestellt wird. Die Grafik wird dabei verzerrt. Sinnvoll ist eine Prozentangabe beispielsweise bei Grafiken, die eine Linie darstellen (vgl. *Bars*, Kapitel 24.2.3), um diese über die gesamte verfügbare Breite zu strecken. Wenn Sie eine Grafik haben, die Sie an anderer Stelle in kleinerer Form wiederholen möchten (zum Beispiel ein Logo, das auf der Einstiegsseite groß angezeigt werden soll und auf den Unterseiten kleiner), brauchen Sie keine zwei Grafiken. Es genügt, wenn Sie, um eine Grafik kleiner darzustellen, die Angaben zu Breite und Höhe proportional verkleinern. Wenn Ihr Logo beispielsweise die Außmaße 300 Pixel breit und 200 Pixel hoch hat, können Sie es halb so groß darstellen, indem Sie beim Einbinden der Grafik

`width=150 height=100` angeben. Sinnvoll ist das aber nur, wenn die große Grafik bereits angezeigt wurde und sich im Cache-Speicher des WWW-Browsers befindet.

#### 7.1.4 Rahmen um Grafiken

Sie können festlegen, dass der Browser einen Rahmen um die Grafik zeichnet. Das entsprechende Attribut ist jedoch als *deprecated* gekennzeichnet und soll künftig aus dem HTML-Standard entfallen. Umrahmung ist auch mit Hilfe von CSS Stylesheets möglich, und dort mit viel mehr Möglichkeiten.



7.3: Über das Attribut `border=` weisen Sie einem Bild einen Rahmen zu.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Baum am Abend</h1>

<p></p>

</body>
</html>
```



Mit dem Attribut `border=` innerhalb der Grafikreferenz wird ein Rahmen um die Grafik definiert (*border = Rand*). Die Angabe erfolgt in Pixeln und bedeutet die Rahmendicke. Die Rahmenfarbe ist diejenige, die Text an der gleichen Stelle haben würde.

### 7.1.5 Namen für Grafiken

Sie können festlegen, dass eine Grafik einen Namen erhält. Dies ist in Verbindung mit JavaScript sinnvoll. Der Name einer Grafik kann zum Beispiel beim JavaScript-Objekt `images` von Bedeutung sein (Kapitel 19.7). Ein Beispiel:

```
<p></p>
```

Mit `name=` vergeben Sie einen Namen für die Grafik. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (`_`), den Bindestrich (`-`), den Doppelpunkt (`:`) oder den Punkt (`.`). Im Hinblick auf JavaScript darf der Name sogar nur Buchstaben, Ziffern und den Unterstrich (`_`) enthalten. Groß- und Kleinschreibung werden bei Sprachen wie JavaScript ebenfalls unterschieden.

### 7.1.6 Langbeschreibung für Grafiken

Wenn Ihnen die Möglichkeit, mit `alt=` einen gewöhnlichen Alternativtext für eine Grafik zu notieren, nicht genügt, können Sie seit HTML 4.0 auch zu einer anderen Stelle oder Datei verweisen, wo sich nähere Textinformation zu der Grafik befindet. Der Browser könnte beispielsweise nur dann einen Verweis zu dieser Stelle anbieten, wenn die Grafik nicht anzeigbar ist. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Alte Zeiten</h1>

<p>

</p>

</body>
</html>
```



7.4: Der Verweis auf die lange Beschreibung ist bei Netscape ziemlich kompliziert zu erreichen.

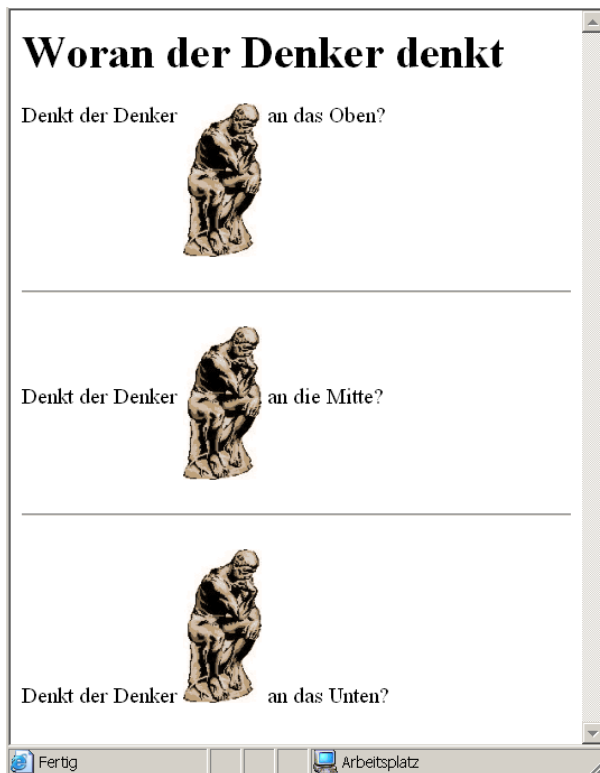
Mit dem Attribut `longdesc=` können Sie auf eine Stelle verweisen, wo die Grafik in Textform näher beschrieben wird. Erlaubt ist ein URI, d.h. bei der Angabe gelten die Regeln zum Referenzieren in HTML.

**Beachten Sie:** Lediglich Netscape 6.0 unterstützt dieses Attribut bislang, und auch nur sehr versteckt: mit der rechten Maustaste über der Grafik klicken, aus dem Kontextmenü die Option *Eigenschaften* wählen, und in dem Dialog, der dann erscheint, auf die Adresse zu *Beschreibung* klicken.

## 7.2 Grafiken ausrichten

### 7.2.1 Grafik zum umgebenden Text ausrichten

Da das `img`-Element ein Inline-Element ist, können Grafiken mitten in einem Text platziert werden. Wenn nun aber die Grafik höher ist als die Zeilenhöhe, dann muss der Text der gleichen Zeile in irgendeiner Weise zur Grafik ausgerichtet werden. Wenn Sie nichts anderes angeben, wird der Text untenbündig zur Grafik ausgerichtet. Sie können jedoch mit Hilfe eines Attributs selbst bestimmen, wie der Text zur Grafik ausgerichtet werden soll. Dieses Attribut ist allerdings als *deprecated* gekennzeichnet und soll künftig aus dem HTML-Standard entfallen. Die gleiche Wirkung lässt sich nämlich auch mit CSS Stylesheets erzielen.



7.5: Der Text kann oben, mittig oder unten an einer Grafik ausgerichtet sein.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Woran der Denker denkt</h1>

<p>Denkt der Denker

an das Oben?</p>
<hr>
<p>Denkt der Denker

an die Mitte?</p>
<hr>
<p>Denkt der Denker
```

```

an das Unten?</p>

</body>
</html>
```

Das Beispiel enthält drei kurze, normalerweise in eine Zeile passende Textabsätze, bei denen jeweils die gleiche Grafik mitten im Text platziert ist. In allen drei Absätzen wird die darin notierte Grafik mit Hilfe von HTML zum Text davor und dahinter ausgerichtet.

Mit dem HTML-Attribut `align` im `<img>`-Tag können Sie Text in der gleichen Zeile zur Grafik ausrichten (*align* = *Ausrichtung*):

- Mit `align="top"` wird der Text obenbündig zur Grafik ausgerichtet (*top* = *oben*).
- Mit `align="middle"` wird der Text mittig zur Grafik ausgerichtet (*middle* = *mittig*).
- Mit `align="bottom"` wird der Text untenbündig zur Grafik ausgerichtet (*bottom* = *unten*).

**Beachten Sie:** Die Browser interpretieren zum Teil noch weitere Angaben. Diese sind jedoch nicht im HTML-Standard verzeichnet und führen deshalb zu ungültigem HTML. Erwähnt seien sie trotzdem:

- Mit `align="texttop"` richten Sie den Text obenbündig zur Grafik aus, und zwar an der Oberkante des kleinsten Textes in der Zeile (*texttop* = *oben am Text*).
- Mit `align="absmiddle"` richten Sie den Text absolut mittig zur Grafik aus, auch bei unterschiedlichen Textgrößen (*absmiddle* = *absolute middle* = *in jedem Fall mittig*).
- Mit `align="absbottom"` richten Sie den Text untenbündig zur Grafik aus, und zwar an der Unterkante des kleinsten Textes in der Zeile (*absbottom* = *absolute bottom* = *in jedem Fall untenbündig*).
- Mit `align="baseline"` richten Sie den Text genauso aus wie mit der Standardangabe `align="bottom"`. Benutzen Sie deshalb besser die Standardangabe.

### 7.2.2 Text um Grafik fließen lassen

Grafiken, die Sie mit dem `<img>`-Tag referenzieren, können Sie links oder rechts ausrichten. Der umgebende Text fließt dabei um die Grafik. Mit zwei weiteren Attributen können Sie Abstand zum umfließenden Text erzeugen, damit der Text nicht direkt an der Grafik »klebt«. Den automatischen Textumfluss können Sie auch abbrechen und die Textfortsetzung unterhalb der Grafik erzwingen. Alle dazu notwendigen Attribute sind allerdings als *deprecated* gekennzeichnet und sollen künftig aus dem HTML-Standard entfallen. Die gleiche Wirkung lässt sich nämlich auch mit CSS Stylesheets erzielen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
```

```

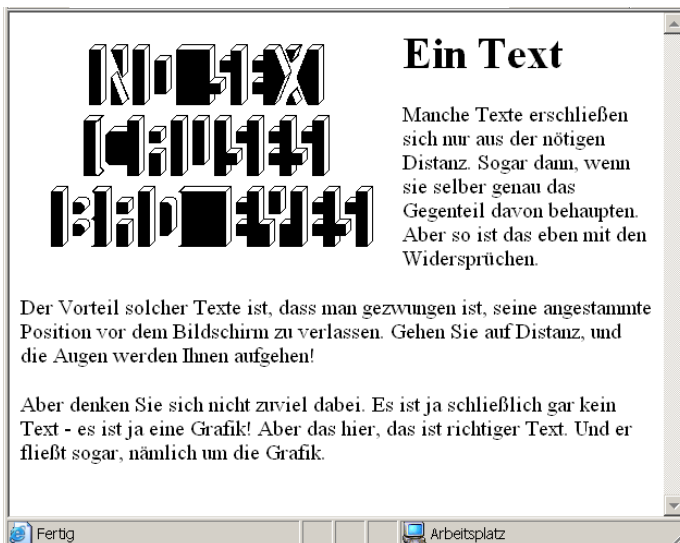
<body>

<h1>
Ein Text
</h1>

<p>Manche Texte erschließen sich nur aus der nötigen Distanz.
<!-- usw. -->
Aber das hier, das ist richtiger Text. Und er fließt sogar, nämlich um
die
Grafik.<br clear="all"></p>

</body>
</html>

```



7.6: Der Text fließt rechts um das Bild.

Mit `align="left"` richten Sie eine Grafik linksbündig aus. Nachfolgend notierter Text fließt rechts um die Grafik. Mit `align="right"` können Sie die Grafik rechtsbündig ausrichten. Der nachfolgend notierte Text fließt dann links um die Grafik. Von »nachfolgend notiertem Text« wird hier nur der Einfachheit halber gesprochen. Es kann sich um beliebige Elemente handeln, also z.B. auch andere Grafiken, Tabellen, Multimedia-Referenzen usw. Um Abstand zwischen Grafik und umfließendem Text zu erzeugen, stehen die Attribute `hspace=` und `vspace=` zur Verfügung:

- Mit `hspace= [Pixel]` bestimmen Sie den Abstand zwischen Grafik und anderen Elementen links bzw. rechts davon (*hspace = horizontal space = horizontaler Abstand*).
- Mit `vspace= [Pixel]` bestimmen Sie den Abstand zwischen Grafik und anderen Elementen darüber bzw. darunter (*vspace = vertical space = vertikaler Abstand*).

Das Attribut `hspace=` betrifft immer den linken **und** den rechten Randabstand von der Grafik, `vspace=` immer den oberen **und** den unteren Randabstand. Wenn Sie also beispielsweise bei

einer links ausgerichteten Grafik `hspace=` definieren, wird nicht nur rechts der Grafik zum nebenstehenden Text ein Abstand erzeugt, sondern auch links zum Rand des Anzeigefensters hin. Um dies zu verhindern, können Sie exakter arbeiten, indem Sie CSS Stylesheets verwenden. Sie können beide Attribute notieren oder auch nur eines davon. Um den automatischen Textumfluss zu unterbrechen und zu erzwingen, dass alles was folgt unterhalb der Grafik angezeigt wird, können Sie einen Zeilenumbruch (Kapitel 5.3) mit dem Attribut `clear=` notieren:

- Mit `<br clear="all">` erreichen Sie, dass der Textfluss ab der nächsten Zeile in jedem Fall unterhalb der Grafik fortgesetzt wird.
- Mit `<br clear="left">` erreichen Sie, dass der Textfluss ab der nächsten Zeile unterhalb einer linksbündig ausgerichteten Grafik fortgesetzt wird.
- Mit `<br clear="right">` erreichen Sie, dass der Textfluss ab der nächsten Zeile unterhalb einer rechtsbündig ausgerichteten Grafik fortgesetzt wird.

### 7.2.3 Grafik und andere Inhalte mit Tabellen ausrichten

In der Praxis kommt es häufig vor, dass Sie Grafiken sauber nebeneinander oder untereinander platzieren und mit Text beschriften wollen. In solchen Fällen bietet sich das Arbeiten mit »blinden«, also unsichtbaren Tabellen an. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

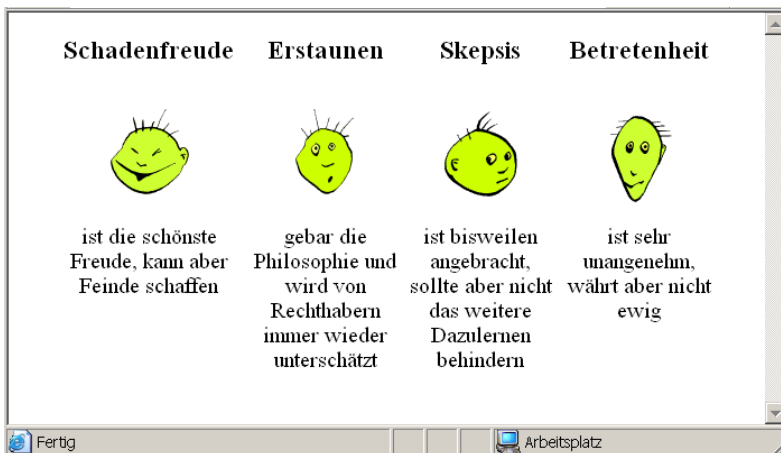
<div align="center">
<table border="0" cellpadding="4" width="600">
  <tr>
    <th><h3>Schadenfreude</h3></th>
    <th><h3>Erstaunen</h3></th>
    <th><h3>Skepsis</h3></th>
    <th><h3>Betretenheit</h3></th>
  </tr><tr>
    <td align="center"></td>
    <td align="center"></td>
    <td align="center"></td>
    <td align="center"></td>
  </tr><tr>
    <td align="center" valign="top">ist die sch&ouml;nste Freude, kann aber Feinde
schaffen</td>
```

```

<td align="center" valign="top">gebar die Philosophie und wird von Rechthabern
immer wieder untersch&auml;tzt</td>
<td align="center" valign="top">ist bisweilen angebracht, sollte aber nicht das
weitere Dazulernen behindern</td>
<td align="center" valign="top">ist sehr unangenehm, w&auml;hrt aber nicht
ewig</td>
</tr>
</table>
</div>

</body>
</html>

```

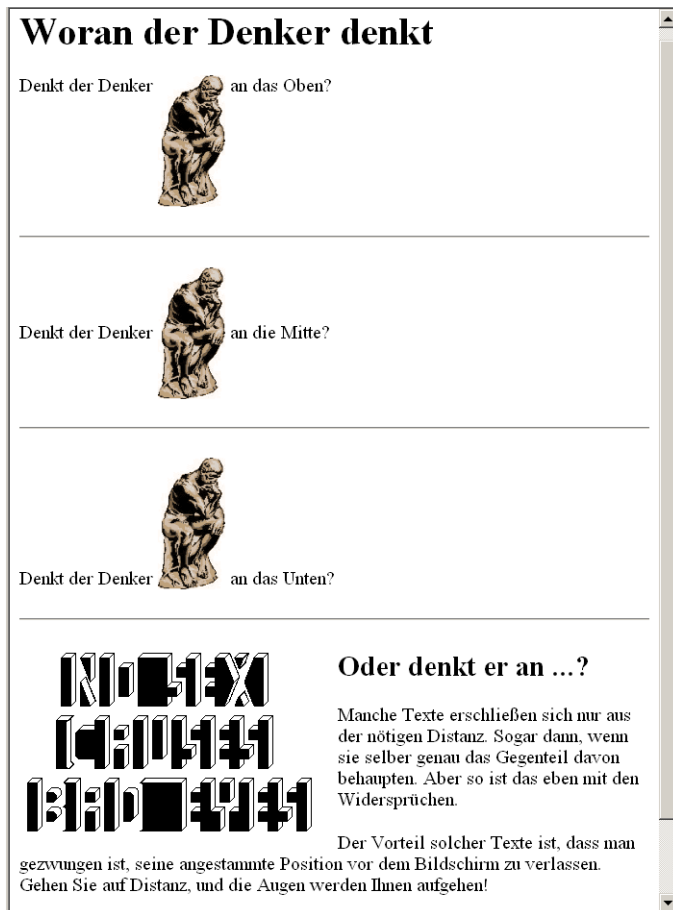


7.7: Eine »unsichtbare« Tabelle richtet Bilder und Text sauber neben- und untereinander aus.

Im Beispiel wird eine blinde Tabelle (`border="0"`) mit vier Spalten definiert. In der ersten Zeile stehen vier Überschriften, in der zweiten vier entsprechende Grafiken und in der dritten vier zugehörige Texte. Die Tabelle ist innerhalb eines allgemeinen Bereichs zentriert ausgerichtet (`<div align="center">...</div>`) und hat eine Breite von 600 Pixeln. Mit `cellpadding="4"` im einleitenden `<table>`-Tag wird sichergestellt, dass sich die Zelleninhalte nicht zu nahe kommen. Mit `align="center"` werden die Datenzellen zentriert ausgerichtet. Da die Beschreibungstexte unterschiedlich lang sein können, wird mit `valign="top"` in deren Datenzellen erreicht, dass sie alle obenbündig ausgerichtet werden.

### 7.2.4 Grafiken ausrichten mit CSS Stylesheets

CSS Stylesheets bieten Eigenschaften an, mit deren Hilfe die beiden Funktionen, Grafik zum umgebenden Text ausrichten und Text um Grafik fließen lassen, ebenso realisierbar sind wie mit HTML. Das »Fein-Tuning« ist mit Hilfe von CSS sogar deutlich genauer. Mit CSS müssen Sie arbeiten, wenn Sie die in diesem Abschnitt beschriebenen Effekte erzielen und dabei aber mit der HTML-Variante »Strict« arbeiten wollen. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: `vertical-align` (Vertikale Ausrichtung), `float` (Textumfluss), `clear` (Fortsetzung bei Textumfluss), Außenrand und Abstand.



7.8: CSS Stylesheets machen die HTML-Attribute zur Ausrichtung von Grafiken überflüssig.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Woran der Denker denkt</h1>

<p>Denkt der Denker

an das Oben?</p>
<hr>
```



```

<p>Denkt der Denker

an die Mitte?</p>
<hr>
<p>Denkt der Denker

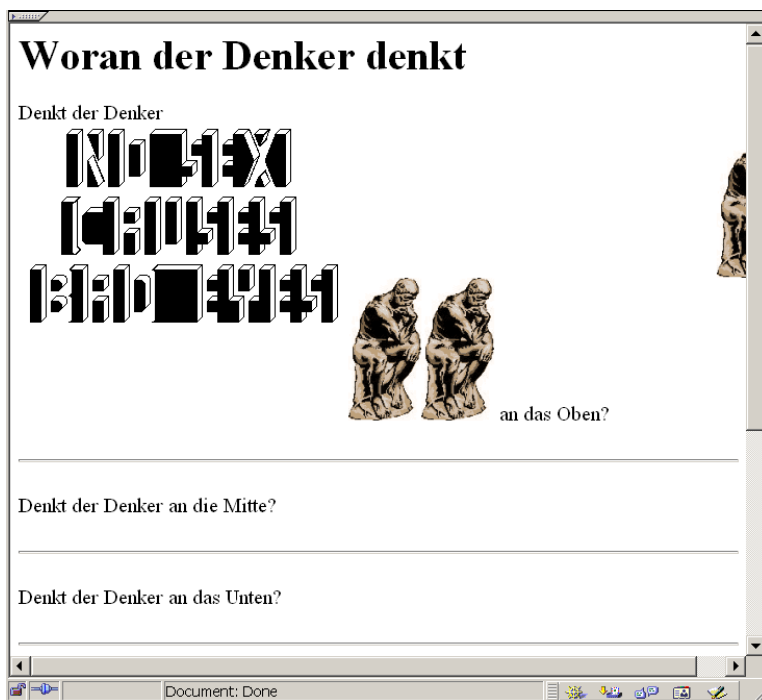
an das Unten?</p>
<hr>

<h2>Oder denkt er an ...?</h2>

<p>Manche Texte erschlie&szlig;en sich nur aus der n&ouml;ltigen Distanz.
<!-- usw. -->
Aber das hier, das ist richtiger Text. Und er flie&szlig;t sogar,
n&auml;mlich um die Grafik.<br style="clear:all"></p>

</body>
</html>

```



7.9: Netscape 4.x steht mit vielen CSS-Angaben auf Kriegsfuß.

Mit `style="vertical-align:text-top"` erreichen Sie in einem `<img>`-Tag das Gleiche wie mit `align="top"`, mit `style="vertical-align:middle"` das Gleiche wie mit `align="middle"`, und mit `style="vertical-align:text-bottom"` das Gleiche wie mit `align="bottom"`.

Mit `style="float:left; margin-right:20px; margin-bottom:10px"` wird erreicht, dass die Grafik links ausgerichtet wird und nachfolgender Text rechts um die Grafik fließt. Dabei wird zwischen Grafik und Text ein Abstand von 20 Pixeln gehalten (links von der Grafik wird dagegen, anders als beim HTML-Attribut `hspace=`, kein Abstand von 20 Pixeln erzeugt). Nach unten hin, also zum Text unterhalb wird ein Abstand von 10 Pixeln definiert.

**Beachten Sie:** Netscape 4.x zerwürfelt die Anzeige dieses Beispiels aufs Übelste. Das ist leider ein typisches Beispiel dafür, wie Netscape 4.x die Entwicklung hin zu standardkonformerem Arbeiten ausbremst.

## 7.3 Grafiken als Verweise

### 7.3.1 Grafiken anstelle von Verweistext definieren

Wenn Sie Verweise setzen, müssen Sie immer auch einen Verweistext definieren, also den Text, der dem Anwender als anklickbar dargestellt wird. Anstelle eines Verweistextes können Sie jedoch auch eine Grafikreferenz notieren. Dann ist die gesamte Grafik anklickbar, und beim Anklicken der Grafik wird der Verweis ausgeführt. Von dieser Möglichkeit wird in der Praxis sehr oft Gebrauch gemacht, bei Werbebannern zum Beispiel, aber auch bei grafischen Navigationsleisten. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<table border="0" cellpadding="0" cellspacing="0"><tr>
<td><a href="home.htm"></a></td>
<td><a href="seite.htm"></a></td>
<td><a href="seite.htm"></a></td>
<td><a href="thema.htm"></a></td>
</tr></table>

<h1>Navigationsleisten</h1>

<p>Was wäre das Web ohne sie!</p>
```

```
</body>
</html>
```

Das Beispiel zeigt eine horizontale Navigationsleiste. Die vier Buttons der Leiste liegen in Form der Grafiken *button1.jpg* bis *button4.jpg* vor. Alle Grafiken sind gleich groß und haben ein einheitliches Aussehen. Sie unterscheiden sich lediglich durch die Beschriftung. Damit die Buttons sauber nebeneinanderliegen und nicht umbrochen werden können, wird eine blinde Tabelle (`border="0"`) dafür notiert. Mit `cellpadding="0"` und `cellspacing="0"` wird in der Tabelle dafür gesorgt, dass die Zellen nahtlos aneinanderliegen und keinen Innenabstand haben. So grenzen die darin enthaltenen Grafiken nahtlos aneinander und erzeugen den Effekt einer Leiste.

Definiert wird eine Zeile mit vier Zellen. In jeder der Zellen wird ein Verweis notiert. Zwischen `<a href="">` und `</a>` wird jedoch kein Verweistext notiert, sondern die Grafikreferenz für je einen der Buttons. Auf diese Weise wird die jeweilige Grafik anklickbar und führt beim Anklicken zu dem Ziel, das bei `href=` notiert ist. Wichtig ist auch die Angabe `border="0"` in den Grafikreferenzen (siehe auch *Rahmen um Grafiken*, Kapitel 7.1.4). Würde diese Angabe fehlen, dann würde der Browser einen Rahmen um die Grafik anzeigen, und zwar in der Farbe für Verweise, um diese als Verweis zu kennzeichnen. Da die Buttons im Beispiel aber für den Anwender eindeutig als Grafiken mit Verweiskennung identifizierbar sind, wird der Rahmen unterdrückt.

Wenn Sie mit `border="0"` den Rahmen um eine als Verweis dienende Grafik unterdrücken, kann der Anwender die Grafik nicht mehr unmittelbar als Verweis erkennen. Nur wenn er mit dem Mauszeiger über die Grafik fährt, kann er an dem Mauszeigersymbol erkennen, dass es sich um einen Verweis handelt. Deshalb sollten Sie den verweiskennzeichnenden Rahmen nur dann unterdrücken, wenn die Grafik auf den ersten Blick als Verweis erkennbar ist.

Wegen der Angabe `border="0"` benutzt das obige Beispiel die HTML-Variante »Transitional«, denn das `border`-Attribut ist im HTML-Standard als *deprecated* gekennzeichnet. Mit einem zentralen CSS-Format `img {border:none}` wäre dieses Problem allerdings elegant zu umgehen, und in allen `<img>`-Tags innerhalb von Verweisen könnte auf das `border`-Attribut verzichtet werden. Bei nicht CSS-fähigen Browsern würde dann aber doch der Rahmen angezeigt.

Nur zum Verständnis: Am obigen Anzeigebeispiel sind in Wirklichkeit vier Dateien beteiligt, die alle den gleichen Aufbau und immer an der gleichen Stelle die Navigationsleiste haben. Nur sind die anklickbaren Grafiken in jeder der Dateien mit unterschiedlichen Verweiszielen verknüpft. Damit zeigt das Beispiel, wie sich mit verschiedenen Dateien, die ein einheitliches Layout und die Navigationsleiste stets an der gleichen Stelle haben, ein in sich geschlossenes Web-Projekt realisieren lässt.

## 7.4 Verweissensitive Grafiken (Image Maps)

### 7.4.1 Verweissensitive Grafiken definieren

Verweissensitive Grafiken sind Grafiken, in denen der Anwender mit der Maus auf ein Detail klicken kann. Daraufhin wird ein Verweis ausgeführt. Auf diese Weise kann der Anwender in einigen Fällen wesentlich intuitiver und schneller zu Information gelangen als durch lange verbale Verweislisten. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Schnell zur Stadt oder Region Ihrer Wahl!</h1>

<map name="Landkarte">
<area shape="rect" coords="11,10,59,29" href="http://www.koblenz.de/"
alt="Koblenz">
<area shape="rect" coords="42,36,96,57" href="http://www.wiesbaden.de/"
alt="Wiesbaden">
<area shape="rect" coords="42,59,78,80" href="http://www.mainz.de/" alt="Mainz">
<area shape="rect" coords="100,26,152,58" href="http://www.frankfurt.de/"
alt="Frankfurt">
<area shape="rect" coords="27,113,93,134" href="http://www.mannheim.de/"
alt="Mannheim">
<area shape="rect" coords="100,138,163,159" href="http://www.heidelberg.de/"
alt="Heidelberg">
<area shape="rect" coords="207,77,266,101" href="http://www.wuerzburg.de/"
alt="W&uuml;rzburg">
<area shape="rect" coords="282,62,344,85" href="http://www.bamberg.de/"
alt="Bamberg">
<area shape="rect" coords="255,132,316,150" href="http://www.nuernberg.de/"
alt="N&uuml;rnnberg">
<area shape="rect" coords="78,182,132,200" href="http://www.karlsruhe.de/"
alt="Karlsruhe">
<area shape="rect" coords="142,169,200,193" href="http://www.heilbronn.de/"
alt="Heilbronn">
<area shape="rect" coords="140,209,198,230" href="http://www.stuttgart.de/"
alt="Stuttgart">
<area shape="rect" coords="187,263,222,281" href="http://www.ulm.de/" alt="Ulm">
<area shape="rect" coords="249,278,304,297" href="http://www.augsburg.de/"
alt="Augsburg">
<area shape="poly" coords="48,311,105,248,96,210,75,205,38,234,8,310"
href="http://www.baden-aktuell.de/" alt="Baden">
</map>

<p></p>

</body>
</html>

```

Mit `<map name="[Name]">` leiten Sie die Definition der verweissensitiven Flächen einer Grafik ein. Beim `name`-Attribut vergeben Sie einen Namen für die verweissensitive Grafik. Dieser Name

muss nichts mit dem Dateinamen der Grafik zu tun haben. Es handelt sich vielmehr um einen Ankernamen, der die gleiche Bedeutung hat wie der Name in einem Anker innerhalb einer HTML-Datei (Kapitel 6.2.2). Vergeben Sie keine zu langen Namen. Namen dürfen keine Leerzeichen und keine deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (\_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Das `map`-Element kann an einer beliebigen Stelle innerhalb des Körpers einer HTML-Datei (also zwischen `<body>` und `</body>`) stehen. Es erzeugt selbst keine Bildschirmausgabe. Es empfiehlt sich jedoch, das Element an einer markanten, gesonderten Stelle innerhalb der Datei zu notieren, z.B. am Anfang oder am Ende des Dateikörpers.

Zwischen dem einleitenden `<map...>` und dem abschließenden `</map>` definieren Sie die verweissensitiven Flächen. Mit `<area...>` definieren Sie einzelne verweissensitive Flächen einer bestimmten Grafik, die Sie an einer anderen Stelle einbinden. Mit `shape="rect"` bestimmen Sie eine viereckige Fläche, mit `shape="circle"` einen Kreis, und mit `shape="polygon"` können Sie ein beliebiges Vieleck als verweissensitiv definieren. Bei `coords=` geben Sie die Koordinaten der verweissensitiven Flächen an. Die Pixelangaben bedeuten absolute Werte innerhalb der Grafik, die verweissensitiv sein soll. Trennen Sie alle Pixelwerte durch Kommata.



7.10: Über ausgewählten Städten verwandelt sich der Mauszeiger in das typische Hand-Symbol, das einen Link anzeigt.

- Ein **Viereck** (`shape="rect"`) definieren Sie mit den Koordinaten für `x1,y1,x2,y2`, wobei bedeuten:  
`x1` = linke obere Ecke, Pixel von links  
`y1` = linke obere Ecke, Pixel von oben

x2 = rechte untere Ecke, Pixel von links

y2 = rechte untere Ecke, Pixel von oben

- Einen **Kreis** (`shape="circle"`) definieren Sie mit den Koordinaten für x,y,r, wobei bedeuten:  
x = Mittelpunkt, Pixel von links  
y = Mittelpunkt, Pixel von oben  
r = Radius in Pixel
- Ein **Polygon** (`shape="polygon"`, Vieleck) definieren Sie mit den Koordinaten x1,y1,x2,y2 ...  
x<sub>n</sub>,y<sub>n</sub>, wobei bedeuten:  
x = Pixel einer Ecke von links  
y = Pixel einer Ecke von oben  
Sie können so viele Ecken definieren wie Sie wollen. Von der letzten definierten Ecke müssen Sie sich eine Linie zur ersten definierten Ecke hinzudenken. Diese schließt das Polygon.

Mit dem Attribut `href=` bestimmen Sie das Verweisziel, das aufgerufen werden soll, wenn der Anwender die verweissensitive Fläche anklickt. Dabei gelten die Regeln zum Referenzieren in HTML – es kann sich also um beliebige Verweisziele innerhalb oder außerhalb des eigenen Web-Projekts handeln. Wenn Sie eine Fläche explizit als nicht anklickbar ausweisen wollen, können Sie dies mit dem Attribut `nohref` tun (ohne Wertzuweisung). Nötig ist dies allerdings nicht, denn Flächen, die nicht durch `href=` abgedeckt sind, sind automatisch nicht anklickbar.

Mit dem `alt`-Attribut notieren Sie Alternativtext für den Fall, dass die verweissensitive Fläche nicht angezeigt werden kann. Dieses Attribut ist ein **Pflichtattribut**, d.h. Sie müssen es notieren, um gültiges HTML zu erzeugen.

Die Grafik, die verweissensitive Flächen haben soll, referenzieren Sie auf herkömmliche Weise mit Hilfe des `<img>`-Tags. Um die Grafik als verweissensitiv zu kennzeichnen, müssen Sie das Attribut `usemap=` notieren. Dieses Attribut erwartet als Wertzuweisung einen URI, der zu der Stelle führt, an der das zugehörige `map`-Element notiert ist. Normalerweise ist dieses Element in der gleichen Datei notiert. Deshalb besteht die Zuweisung einfach in einem Gatterzeichen `#` und dem Namen des Ankers, der bei `<map name=` definiert wurde.

**Beachten Sie:** Um die gewünschten Pixelkoordinaten für verweissensitive Flächen einer Grafik zu erhalten, können Sie beispielsweise ein Grafikprogramm benutzen, bei dem Sie mit der Maus in der angezeigten Grafik herumfahren können und dabei die genauen Pixelkoordinaten des Mauszeigers angezeigt bekommen.

Um beim Überfahren einer verweissensitiven Fläche mit der Maus ein kleines Fenster (»Tooltip«-Fenster) anzuzeigen, können Sie in den `<area>`-Tags jeweils das Universalattribut `title=` "Mein angezeigter Text" verwenden. Dies wird allerdings nicht von allen Browsern interpretiert.

Innerhalb eines jeden `<area>`-Tags ist auch das Attribut `tabindex=` erlaubt. Die Wirkung ist die gleiche wie bei der Tabulator-Reihenfolge bei Verweisen (Kapitel 6.7) und wird dort näher beschrieben. Ebenfalls erlaubt ist das Attribut `accesskey=`. Dies funktioniert genau wie bei Tastaturkürzeln für Verweise und wird in Kapitel 6.7 näher beschrieben.

Wenn Sie **XHTML-konform** arbeiten, müssen Sie das `area`-Element als inhaltsleer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<area.../>`. Weitere Informationen dazu im Kapitel 14.1.11 über *XHTML und HTML*.

Im `<area>`-Tag kommen häufig auch JavaScript-Event-Handler zum Einsatz. Beim Aufruf von JavaScript-Funktionen mit Event-Handleern darf jedoch im `<area>`-Tag das `href`-Attribut nicht fehlen, sonst streikt der Netscape Navigator. Um das Problem zu lösen, können Sie eine Angabe notieren wie:

```
<area shape="rect" coords="1,1,249,49" href="#" onclick="IhreFunktion()" alt="Kurze Beschreibung">
```

Durch die Wertzuweisung `"#"` an das `href`-Attribut wird ein »leerer« Verweis erzeugt, und Netscape führt dann auch den Event-Handler (hier: `onClick=`) aus.

### 7.4.2 Server-seitige verweissensitive Grafiken

Bei dieser Methode ist zum Verwalten von verweissensitiven Grafiken eine Kommunikation zwischen Web-Browser und Web-Server erforderlich. Der Nachteil ist, dass solche verweissensitive Grafiken nicht lokal in jeder Umgebung funktionieren, und dass der Server zusätzlich belastet wird. Insgesamt kann diese Methode heute als veraltet gelten und wird hier nur noch der Vollständigkeit halber erwähnt.

Das Prinzip funktioniert folgendermaßen: Aufgrund eines HTML-Attributs weiß der Web-Browser, dass es sich bei einer Grafik um eine verweissensitive Grafik handelt. Wenn der Anwender dann irgendwo auf die angezeigte Grafik klickt, überträgt der Web-Browser die Pixelkoordinaten des Mausklicks (relativ zur oberen linken Ecke der Grafik) an eine spezielle »Imagemap«-Software auf dem Server-Rechner. Diese Software ermittelt, ob die Pixelkoordinaten innerhalb einer als verweissensitiv definierten Fläche der Grafik liegen. Wenn ja, ermittelt die Software, welcher Verweis dieser sensitiven Fläche zugeordnet ist. Die entsprechende Verweisadresse wird dem Web-Browser gesendet. Dieser führt den Verweis dann aus.

Wenn Sie eine verweissensitive Grafik nach diesem Schema einsetzen möchten, fragen Sie bei Ihrem Provider nach, welche Software zum Verarbeiten verweissensitiver Grafiken auf diesem Server im Einsatz ist und was genau Sie brauchen. In der Regel brauchen Sie eine so genannte »Map-Datei« und einen Eintrag in einer Konfigurationsdatei. In der Map-Datei ordnen Sie verweissensitive Flächen und gewünschte Verweise einander zu. In der Konfigurationsdatei ist ein Eintrag für die Server-Software nötig.

Bei der Syntax der Map-Dateien gibt es mehrere Varianten. Fragen Sie gegebenenfalls Ihren Provider nach Beispieldateien und nach einer Dokumentation zu der von ihm angebotenen Variante. Das folgende Beispiel zeigt den Teil, den Sie in der HTML-Datei notieren müssen.

```
<a href="MapName"></a>
```

Es handelt sich also um eine schlichte Grafik als Verweis. Im Beispiel wird die Datei `datei.gif` auf diese Weise notiert. Das `<img>`-Tag muss das zusätzliche Attribut `ismap` enthalten. Hierdurch wird dem Server-Rechner mitgeteilt, dass es sich um eine verweissensitive Grafik handelt. Das Ziel des Verweises bei `href=` ist keine Datei, sondern ein symbolischer Name, der zuvor in der oben angesprochenen Konfigurationsdatei des Server-Rechners vergeben wurde. Im Beispiel ist das der symbolische Name `"MapName"`.

## 8 Tabellen

### 8.1 Aufbau einer Tabelle

#### 8.1.1 Tabellen definieren

Sie können in HTML Tabellen definieren, um tabellarische Daten darzustellen oder um Text und Grafik attraktiver am Bildschirm zu verteilen. Obwohl Tabellen natürlich vornehmlich zur Darstellung tabellarischer Daten geschaffen wurden, sind sie in der heutigen Praxis des Web-Designs vor allem als Grundgestaltungsmittel für Seitenlayouts nicht mehr wegzudenken. Mehr dazu im Abschnitt 8.6 *Tabellen als Mittel für Web-Seitenlayouts*. Puristen, die HTML so weit wie möglich von gestalterischen Aufgaben befreit sehen wollen, rümpfen darüber zwar die Nase, aber so klar sind die Grenzen zwischen Struktur und Gestaltung nicht immer. Die saubere Verteilung von Information auf einer Web-Seite ist durchaus ein Aspekt der Strukturierung, obwohl sie zugleich ein Aspekt der Gestaltung ist. Rein optisch lässt sich also grundsätzlich unterscheiden zwischen Tabellen, die Gitternetzlinien haben (für tabellarische Daten), und Tabellen ohne sichtbare Gitternetzlinien (so genannte »blinde Tabellen« für mehrspaltigen Text oder für saubere Verteilung von Inhalten auf einer Web-Seite).

### Tabelle mit Gitternetzlinien

| Berlin   | Hamburg     | München          |
|----------|-------------|------------------|
| Miljöh   | Kiez        | Bierdampf        |
| Buletten | Frikadellen | Fleischpflanzerl |

### Tabelle ohne Gitternetzlinien (blinde Tabelle)

ARQ

Automatic Repeat Request. Eine allgemeine Bezeichnung für Fehlerprotokolle, die Übertragungsfehler erkennt und defekte Blöcke selbständig wiederholt

HDLC

High Level Data Link Control. Ein Standard-Protokoll, das von der Kommission für internationale Standards für Softwareanwendungen in synchronen Anlagen verwendet wird.

8.1: Mit Tabellen präsentieren Sie nicht nur Daten, sondern gestalten das Layout von Web-Seiten.



Das zum Bild gehörige Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Aufbau einer Tabelle</title>
</head>
<body>

<h1>Tabelle mit Gitternetzlinien</h1>

<table border="1">
<tr>
<th>Berlin</th>
<th>Hamburg</th>
<th>München</th>
</tr><tr>
<td>Miljö</td>
<td>Kiez</td>
<td>Bierdampf</td>
</tr><tr>
<td>Buletten</td>
<td>Frikadellen</td>
<td>Fleischpflanzerl</td>
</tr>
</table>

<h1>Tabelle ohne Gitternetzlinien (blinde Tabelle)</h1>

<table border="0">
<tr>
<td><h2>ARQ</h2></td>
<td><p>Automatic Repeat Request. Eine allgemeine Bezeichnung für
Fehlerprotokolle, die Übertragungsfehler erkennt und defekte Blöcke
selbständig wiederholt</p></td>
</tr><tr>
<td><h2>HDLC</h2></td>
<td><p>High Level Data Link Control. Ein Standard-Protokoll, das von der Kommission
für internationale Standards für Softwareanwendungen in synchronen
Anlagen verwendet wird.</p></td>
</tr>
</table>

</body>
</html>
```

|         |      |      |      |          |
|---------|------|------|------|----------|
| <table> |      |      |      |          |
| <tr>    | <th> | <th> | <th> | </tr>    |
| <tr>    | <td> | <td> | <td> | </tr>    |
| <tr>    | <td> | <td> | <td> | </tr>    |
|         |      |      |      | </table> |

8.2: Die Grafik zeigt die Wirkung der HTML-Elemente, die eine Tabelle erzeugen.

`<table>` leitet eine Tabelle ein (*table* = *Tabelle*). Wenn die Tabelle sichtbare Gitternetzlinien enthalten soll, müssen Sie im einleitenden `<table>`-Tag das Attribut `border=` notieren und ihm einen Wert größer 0 zuweisen. Der angegebene Wert ist dann die Breite des Rahmens in Pixeln. Um eine blinde Tabelle ohne sichtbaren Rahmen und Gitternetzlinien zu erzeugen, lassen Sie die Angabe zu `border=` entweder weg oder – was sauberer ist – Sie notieren `border="0"`.

`<tr>` leitet eine neue Tabellenzeile ein (*tr* = *table row* = *Tabellenzeile*). Im Anschluss daran werden die Zellen (Spalten) der betreffenden Reihe definiert. Am Ende einer Tabellenzeile wird ein abschließendes Tag `</tr>` notiert.

Eine Tabelle kann Kopfzellen und gewöhnliche Datenzellen enthalten. Text in Kopfzellen wird hervorgehoben (meist fett und zentriert ausgerichtet). `<th>` leitet eine Kopfzeile ein, `<td>` eine normale Datenzeile (*th* = *table header* = *Tabellenkopf*, *td* = *table data* = *Tabellendaten*). Der Inhalt einer Zelle wird jeweils hinter dem Tag notiert. Obwohl die zugehörigen End-Tags `</th>` bzw. `</td>` offiziell optional sind, ist dringend zu empfehlen, sie immer und in jedem Fall zu notieren.

In einer Tabellenzeile können beliebige Elemente stehen, d.h. außer normalem Text z.B. auch andere Block- und Inline-Elemente. Sogar eine weitere Tabelle können Sie innerhalb einer Zelle definieren.

**Beachten Sie:** Die Anzahl der Zellen sollte bei jeder Zeile gleich sein, sodass die Tabelle durchweg die gleiche Anzahl Spalten pro Zeile hat. In der ersten Zeile, die Sie definieren, legen Sie deshalb durch die Anzahl der dort definierten Zellen die Anzahl der Spalten Ihrer Tabelle fest.

Tabellenzellen dürfen auch leer sein. Wenn Sie in einer Zeile für eine Spalte keine Daten eingeben wollen, notieren Sie ein einfaches `<td></td>`. Beachten Sie dabei jedoch, dass viele Web-Browser die Zelle in diesem Fall als »nicht vorhanden« darstellt. Probieren Sie deshalb auch mal die Notation `<td>&#160;</td>` für leere Tabellenzellen aus.

### 8.1.2 Spalten vordefinieren

Die Darstellung einer Tabelle ergibt sich zwar automatisch aus den definierten Zeilen und Spalten. Doch für einen Web-Browser ist es nicht ganz einfach, die Darstellung frühzeitig zu ermitteln. Er muss erst die gesamte Tabelle einlesen, bevor er irgendetwas davon darstellen kann. Bei großen Tabellen kann das zu unschönen leeren Bildschirmhalten während des Seitenaufbaus führen.

HTML 4.0 bietet eine Syntax an, um dem Browser gleich zu Beginn der Tabelle mitzuteilen, wie viele Spalten die Tabelle hat und wie breit diese sind. Dadurch kann der Browser die Tabelle schneller aufbauen, d.h. bereits Teile der Tabelle anzeigen, bevor die gesamte Tabelle eingelesen ist. Ältere Browser ignorieren diese Angaben allerdings.

|                         |                        |                     |
|-------------------------|------------------------|---------------------|
| <b>Beispielschema 1</b> |                        |                     |
| 1. Zeile,<br>1. Spalte  | 1. Zeile, 2.<br>Spalte | 1. Zeile, 3. Spalte |
| <b>Beispielschema 2</b> |                        |                     |
| 1. Zeile, 1. Spalte     | 1. Zeile, 2. Spalte    | 1. Zeile, 3. Spalte |
| <b>Beispielschema 3</b> |                        |                     |
| 1. Zeile, 1. Spalte     | 1. Zeile, 2. Spalte    | 1. Zeile, 3. Spalte |

8.3: Die Darstellung einer Tabelle für die drei folgenden Schemata.

#### Beispielschema 1:

```
<table border="1">
  <colgroup>
    <col width="80">
    <col width="100">
    <col width="320">
  </colgroup>
  <tr>
    <td>1. Zeile, 1. Spalte</td>
    <td>1. Zeile, 2. Spalte</td>
    <td>1. Zeile, 3. Spalte</td>
  </tr>
  <!-- usw. andere Zeilen der Tabelle -->
</table>
```

#### Beispielschema 2:

```
<table border="1">
  <colgroup width="200" span="3">
  </colgroup>
  <tr>
    <td>1. Zeile, 1. Spalte</td>
    <td>1. Zeile, 2. Spalte</td>
    <td>1. Zeile, 3. Spalte</td>
  </tr>
  <!-- usw. andere Zeilen der Tabelle -->
</table>
```

## Beispielschema 3:

```
<table border="1" width="100%">
  <colgroup>
    <col width="4*">
    <col width="2*">
    <col width="1*">
  </colgroup>
  <tr>
    <td>1. Zeile, 1. Spalte</td>
    <td>1. Zeile, 2. Spalte</td>
    <td>1. Zeile, 3. Spalte</td>
  </tr>
  <!-- usw. andere Zeilen der Tabelle -->
</table>
```

Mit `<colgroup>` leiten Sie hinter dem einleitenden `<table>`-Tag eine Vorab-Definition der Tabellenspalten ein (*colgroup* = *column group* = *Spaltengruppe*). Dabei haben Sie zwei Möglichkeiten: Entweder Sie möchten unterschiedlich breite Tabellenspalten haben. Dann gehen Sie so vor wie im obigen Beispielschema 1. Oder Sie haben eine Tabelle, in der alle Spalten die gleiche einheitliche Breite haben sollen. Dann können Sie so vorgehen wie im obigen Beispielschema 2.

Im Beispielschema 1 enthält das `<colgroup>`-Tag keine weiteren Attribute. Dafür notieren Sie im Anschluss an `<colgroup>` für jede einzelne gewünschte Tabellenspalte je ein `<col>`-Tag. Das erste `<col>`-Tag definiert die erste Spalte, das zweite die zweite Spalte usw. Wenn Sie keine weiteren Angaben machen, wird die Breite der Spalten automatisch aufgrund des Tabelleninhalts ermittelt. Mit `width= [Pixel/Prozent]` können Sie jedoch eine Spaltenbreite für die einzelnen Spalten vorgeben (*width* = *Breite*). Mit `width="100"` erzwingen Sie beispielsweise eine Spaltenbreite von 100 Pixeln, und mit `width=33%` eine Breite von einem Drittel der Breite der Gesamttabelle.

Im Beispielschema 2 werden keine `<col>`-Tags notiert. Stattdessen notieren Sie im einleitenden `<colgroup>`-Tag das Attribut `span=` (*span* = *spannen*). Als Wert weisen Sie die Anzahl der Spalten zu, die Sie wünschen. Mit dem Attribut `width=` können Sie in diesem Fall eine einheitliche Spaltenbreite für alle Spalten definieren.

Bei `width=` haben Sie neben der Möglichkeit, Pixel oder Prozentwerte anzugeben, auch noch eine dritte Möglichkeit: Sie können das relative Breitenverhältnis der Spalten untereinander bestimmen, unabhängig davon, wie breit die Tabelle im Verhältnis zum Anzeigefenster ist. Eine solche Möglichkeit stellt das obige Beispielschema 3 vor. Bei Breitenangaben dieser Art weisen Sie `width=` eine Zahl und dahinter ein Stern-Zeichen zu. Das Stern-Zeichen ist dabei nur ein Signalzeichen für den Browser, dass er die Zahlen davor nicht als Pixel interpretieren soll. Wichtig sind die Zahlen. Im obigen Beispiel 3 werden drei Spalten definiert, bei denen die relativen Zahlen 4, 2 und 1 in der Summe 7 ergeben. Damit definieren Sie eine Tabelle, bei der die erste Spalte vier Siebtel der Tabellenbreite einnimmt, die zweite Spalte zwei Siebtel und die dritte Spalte ein Siebtel.

Zur Geltung kommt ein relatives Spaltenverhältnis aber erst, wenn Sie außerdem eine Breite für die gesamte Tabelle angeben. Im obigen Beispielschema 3 geschieht das durch die Angabe `width="100%"` im einleitenden `<table>`-Tag.

**Beachten Sie:** `<col>`-Tags dürfen kein Abschluss-Tag haben. Das abschließende `</colgroup>`-Tag ist dagegen optional. Notieren Sie, wenn Sie XHTML-konform arbeiten wollen, das `<col>`-Tag beispielsweise so: `<col width="4*" />`.

Auch das `<col>`-Tag darf das Attribut `span=` erhalten. Dadurch gruppieren Sie jedoch nicht mehrere Spalten zu einer, sondern Sie geben lediglich an, dass Attribute dieser Spalte auch für die nächsten soundsoviel Spalten gelten sollen. Wenn Sie beispielsweise `<col span="3" width="100">` notieren, erhalten diese und die nächsten zwei Spalten die Breite 100 Pixeln.

Das `<colgroup>`-Tag darf die Attribute `span=` und `width=` auch dann enthalten, wenn unterhalb davon `<col>`-Tags definiert werden. Dabei überschreibt die Anzahl der definierten `<col>`-Tags jedoch die Angabe, die mit `<colgroup span=` gemacht wurde, und das Attribut `width=` innerhalb eines `<col>`-Tags hat Vorrang vor der Angabe `width=` im `<colgroup>`-Tag.

Es ist auch erlaubt, mehrere `<colgroup>`-Tags zu notieren. So können Sie beispielsweise mit `<colgroup width="100" span="3">` und `<colgroup width="50" span="5">` hintereinander notiert insgesamt 8 Spalten für die Tabelle definieren, wobei die ersten drei Spalten eine Breite von 100 Pixeln erhalten und die nachfolgenden fünf Spalten eine Breite von 50 Pixeln.

Für die Breite der Tabellenspalten gelten die gleichen Hinweise wie beim Erzwingen von Spaltenbreiten (Kapitel 8.2.4) auf herkömmliche Weise.

### 8.1.3 Kopf, Körper und Fuß einer Tabelle definieren

Sie können eine Tabelle logisch in Bereiche aufteilen: einen Kopfbereich, einen oder mehrere Datenbereiche und einen Fußbereich. Interessant ist dies zum Beispiel in Verbindung mit der Möglichkeit, Regeln für Gitternetzlinien (Kapitel 8.2.3) zu bestimmen. Beim Ausdruck langer Tabellen sollte der Browser Tabellenkopf und Tabellenfuß auf jeder Seite wiederholen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Spalten vordefinieren</title>
</head>
<body>

<h1>Betroffene Menschen</h1>

<table border="1" rules="groups">
  <thead>
    <tr>
      <th>Assoziation 1</th>
      <th>Assoziation 2</th>
      <th>Assoziation 3</th>
```

```

</tr>
</thead>
<tfoot>
<tr>
  <td><i>betroffen:<br>4 Mio. Menschen</i></td>
  <td><i>betroffen:<br>2 Mio. Menschen</i></td>
  <td><i>betroffen:<br>1 Mio. Menschen</i></td>
</tr>
</tfoot>
<tbody>
<tr>
  <td>Berlin</td>
  <td>Hamburg</td>
  <td>München</td>
</tr><tr>
  <td>Miljö</td>
  <td>Kiez</td>
  <td>Bierdampf</td>
</tr><tr>
  <td>Buletten</td>
  <td>Frikadellen</td>
  <td>Fleischpflanzerl</td>
</tr>
</tbody>
</table>

</body>
</html>

```

| Assoziation 1                         | Assoziation 2                         | Assoziation 3                         |
|---------------------------------------|---------------------------------------|---------------------------------------|
| Berlin                                | Hamburg                               | München                               |
| Miljö                                 | Kiez                                  | Bierdampf                             |
| Buletten                              | Frikadellen                           | Fleischpflanzerl                      |
| <i>betroffen:<br/>4 Mio. Menschen</i> | <i>betroffen:<br/>2 Mio. Menschen</i> | <i>betroffen:<br/>1 Mio. Menschen</i> |

8.4: Der Browser ordnet Kopf und Fuß einer Tabelle automatisch an.

Den Tabellenkopf leiten Sie mit `<thead>` ein (*thead* = *table head* = *Tabellenkopf*). Daran anschließend können Sie eine oder mehrere Zeilen der Tabelle notieren, die zum Kopfbereich gehören sollen. Mit `</thead>` schließen Sie den Tabellenkopf ab (das End-Tag ist bei allen

Elementen für Tabellenkopf, Tabellenfuß und Tabellenkörper optional, aber dringend zu empfehlen).

Bevor Sie einen oder mehrere Tabellenkörper-Elemente notieren, müssen Sie hinter dem Tabellenkopf den Tabellenfuß notieren. Diesen leiten Sie mit `<tfoot>` ein (*tfoot* = *table foot* = *Tabellenfuß*). Daran anschließend können Sie eine oder mehrere Zeilen der Tabelle notieren, die zum Fußbereich gehören sollen. Mit `</tfoot>` schließen Sie den Tabellenfuß ab.

Einen Tabellenkörper leiten Sie mit `<tbody>` ein (*tbody* = *table body* = *Tabellenkörper*). Daran anschließend notieren Sie den Datenbereich mit einer oder mehreren Tabellenzeilen. Mit `</tbody>` schließen Sie den Tabellenkörper ab.

**Beachten Sie:** Wenn Sie mit den Elementen `thead`, `tfoot` und `tbody` arbeiten, müssen Sie immer alle drei Elemente verwenden, und zwar immer in der Reihenfolge `thead` -> `tfoot` -> `tbody`. Die Elemente `thead` und `tfoot` dürfen pro Tabelle nur einmal vorkommen, das `tbody`-Element einmal oder beliebig oft.

## 8.2 Gestaltung einer Tabelle

### 8.2.1 Zellenabstand und Zelleninnenabstand

Sie können den Abstand zwischen den Zeilen und Spalten einer Tabelle in Pixeln bestimmen. Ferner können Sie den Abstand zwischen Zellenrand und Zelleninhalt bestimmen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Spalten vordefinieren</title>
</head>
<body>

<h1>Weit auseinander</h1>

<table border="8" cellspacing="10" cellpadding="20">
  <tr>
    <th>Berlin</th>
    <th>Hamburg</th>
    <th>M&uuml;nchen</th>
  </tr>
  <tr>
    <td>Milj&ouml;h</td>
    <td>Kiez</td>
    <td>Bierdampf</td>
  </tr>
  <tr>
    <td>Buletten</td>
    <td>Frikadellen</td>
```

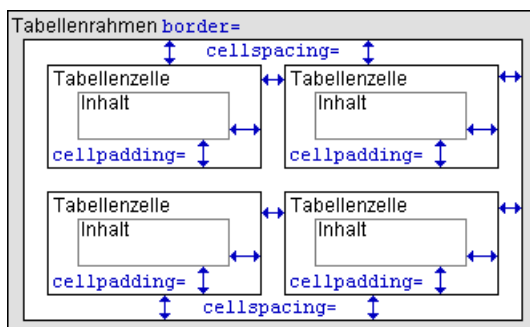
```

        <td>Fleischpflanzerl</td>
    </tr>
</table>
</body>
</html>

```



8.5: Der Abstand zwischen den Zellen und zwischen Zellinhalt und Rand ist einstellbar.



8.6: Die Grafik zeigt die Wirkungsweise der Attribute im einleitenden `<table>`-Tag.

Mit `cellpadding=` bestimmen Sie den Innenabstand einer Zelle, also den Abstand zwischen Zellenrand und Zellinhalt in Pixeln (*cellpadding* = *Zelleninnenabstand*). Mit `cellspacing=` bestimmen Sie den Abstand der Zellen untereinander in Pixeln (*cellspacing* = *Zellenabstand*). Wenn Sie mit `border=` einen sichtbaren Rahmen und sichtbare Gitternetzlinien in der Tabelle erzeugen, ist die Wirkungsweise von `cellpadding=` und `cellspacing=` besser nachvollziehbar. Bei blinden Tabellen erzeugen beide Attribute optisch gesehen einfach nur »Abstand«.



## 8.2.2 Regeln für den Außenrahmen

Wenn Sie mit `border=` einen sichtbaren Tabellenrahmen erzeugen, erhalten automatisch alle Seiten einen Rahmen. Sie können aber auch genau bestimmen, welche Seiten eines Außenrahmens angezeigt werden und welche nicht.



| Assoziation 1 | Assoziation 2 | Assoziation 3    |
|---------------|---------------|------------------|
| Berlin        | Hamburg       | München          |
| Miljö         | Kiez          | Bierdampf        |
| Buletten      | Frikadellen   | Fleischpflanzerl |

8.7: Eine Tabelle mit ausgeschaltetem Außenrahmen.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Rahmenvereinbarung</h1>

<table border="3" frame="void">
  <tr>
    <td><b>Assoziation 1</b></td>
    <td><b>Assoziation 2</b></td>
    <td><b>Assoziation 3</b></td>
  </tr><tr>
    <td>Berlin</td>
    <td>Hamburg</td>
    <td>München</td>
  </tr><tr>
    <td>Miljö</td>
    <td>Kiez</td>
    <td>Bierdampf</td>
  </tr><tr>
    <td>Buletten</td>
    <td>Frikadellen</td>
    <td>Fleischpflanzerl</td>
  </tr>
```

```

</table>

</body>
</html>

```

Voraussetzung für all diese Angaben ist das Attribut `border=` im einleitenden `<table>`-Tag. Dadurch wird ein Außenrahmen angezeigt. Mit dem Attribut `frame=` können Sie dann bestimmen, an welchen Seiten der Tabellenrahmen gezogen werden soll (*frame = Rahmen*). Folgende andere Angaben sind möglich:

- Mit `frame="box"` (*box = Rechteck*) erreichen Sie, dass der Tabellenrahmen oben, links, rechts und unten sichtbar dargestellt wird (die Angabe ist identisch mit dem, was die Angabe `border=` bewirkt – auch `frame="border"` ist erlaubt und bewirkt das Gleiche).
- Mit `frame="void"` (*void = nichts*) wird überhaupt kein Tabellenrahmen angezeigt. Wenn Sie `border=` angeben, werden jedoch die Gitternetzlinien der Tabelle sichtbar angezeigt. Die Tabelle sieht dann also aus wie ein an allen Seiten offenes Gitter (das obige Beispiel benutzt diese Variante).
- Mit `frame="above"` (*above = oberhalb*) wird nur am oberen Rand der Tabelle eine Rahmenlinie angezeigt.
- Mit `frame="below"` (*below = unterhalb*) wird nur am unteren Rand der Tabelle eine Rahmenlinie angezeigt.
- Mit `frame="hsides"` (*hsides = horizontal sides = horizontale Seiten*) wird nur am oberen und am unteren Rand der Tabelle eine Rahmenlinie angezeigt.
- Mit `frame="vsides"` (*vsides = vertical sides = vertikale Seiten*) wird nur am linken und am rechten Rand der Tabelle eine Rahmenlinie angezeigt.
- Mit `frame="lhs"` (*lhs = left hand side = links*) wird nur am linken Rand der Tabelle eine Rahmenlinie angezeigt.
- Mit `frame="rhs"` (*rhs = right hand side = rechts*) wird nur am rechten Rand der Tabelle eine Rahmenlinie angezeigt.

**Beachten Sie:** Netscape 4.x interpretiert diese Angaben noch nicht.

### 8.2.3 Regeln für Gitternetzlinien

Sie können Regeln aufstellen, welche Gitternetzlinien einer Tabelle angezeigt werden sollen und welche nicht. Ein Beispiel:

```

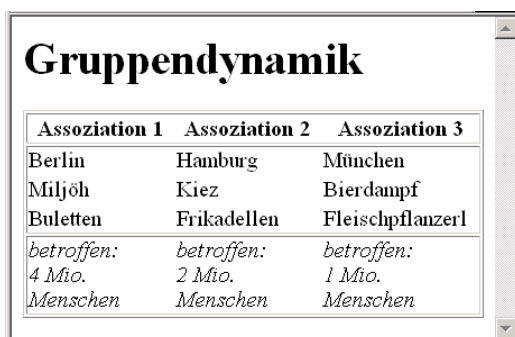
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Gruppendynamik</h1>

```

```

<table border="1" rules="groups">
  <thead>
    <tr>
      <th>Assoziation 1</th>
      <th>Assoziation 2</th>
      <th>Assoziation 3</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td><i>betroffen:<br>4 Mio. Menschen</i></td>
      <td><i>betroffen:<br>2 Mio. Menschen</i></td>
      <td><i>betroffen:<br>1 Mio. Menschen</i></td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Berlin</td>
      <td>Hamburg</td>
      <td>München</td>
    </tr><tr>
      <td>Miljö</td>
      <td>Kiez</td>
      <td>Bierdampf</td>
    </tr><tr>
      <td>Buletten</td>
      <td>Frikadellen</td>
      <td>Fleischpflanzerl</td>
    </tr>
  </tbody>
</table>
</body>
</html>

```



| Assoziation 1                             | Assoziation 2                             | Assoziation 3                             |
|-------------------------------------------|-------------------------------------------|-------------------------------------------|
| Berlin                                    | Hamburg                                   | München                                   |
| Miljö                                     | Kiez                                      | Bierdampf                                 |
| Buletten                                  | Frikadellen                               | Fleischpflanzerl                          |
| <i>betroffen:<br/>4 Mio.<br/>Menschen</i> | <i>betroffen:<br/>2 Mio.<br/>Menschen</i> | <i>betroffen:<br/>1 Mio.<br/>Menschen</i> |

8.8: Eine Tabelle ohne senkrechte Trennlinien zwischen den Zellen.

Voraussetzung für all diese Angaben ist das Attribut `border=` im einleitenden `<table>`-Tag. Dadurch werden Gitternetzlinien angezeigt. Mit `rules=` können Sie im einleitenden `<table>`-Tag Regeln für die Gitternetzlinien bestimmen (*rules = Linien*). Folgende Angaben sind dabei möglich:

- Mit `rules="none"` (*none = keine*) werden überhaupt keine Linien gezogen, der Außenrahmen der Tabelle wird jedoch angezeigt.
- Mit `rules="rows"` (*rows = Reihen*) werden Linien zwischen allen Tabellenzeilen gezogen, nicht jedoch zwischen den Spalten der Tabelle.
- Mit `rules="cols"` (*cols = columns = Spalten*) werden Linien zwischen allen Tabellenspalten gezogen, nicht jedoch zwischen den Zeilen der Tabelle.
- Mit `rules="groups"` (*groups = Gruppen*) werden Linien zwischen Kopf, Körper und Fuß einer Tabelle gezogen (siehe *Kopf, Körper und Fuß einer Tabelle definieren* in Kapitel 8.1.3 – das obige Beispiel verwendet diese Angabe).
- Mit `rules="all"` (*all = alle*) werden Linien zwischen allen Tabellenzellen gezogen (Voreinstellung).

**Beachten Sie:** Netscape 4.x interpretiert diese Angaben noch nicht.

### 8.2.4 Breiten- und Höhenangaben

Sie können für die gesamte Tabelle sowie für einzelne Spalten und Zeilen geeignete Angaben zu Breite und Höhe notieren. Dadurch schaffen Sie »Raum«. Breitenangaben werden durch das Attribut `width=` notiert, Höhenangaben durch das Attribut `height=` (*width = Breite, height = Höhe*). Bei allen Angaben sind absolute Zahlenangaben erlaubt, die die Breite bzw. Höhe als Anzahl Pixel bestimmen, sowie prozentuale Angaben, die am Ende ein Prozentzeichen (%) haben.



8.9: Breite und Höhe von Tabellenzellen lassen sich mit Attributen und CSS steuern.

Die einzige von allen Angaben, die nach der HTML-Variante »Strict« erlaubt ist, ist die Gesamtbreite der Tabelle. Andere Angaben sind als *deprecated* gekennzeichnet und sollen künftig aus dem HTML-Standard entfallen. Solche Angaben sind auch mit Hilfe von CSS Stylesheets möglich. Und schließlich gibt es auch noch Angaben, die zwar von einigen Browsern interpretiert werden, aber nicht mal zum HTML-Standard in der Variante »Transitional« gehören. Die folgende Tabelle listet auf, welche Angaben möglich sind, und welchen Status die Angabe in Bezug auf HTML-Standard und Browser-Kompatibilität hat.

| Angabe (Notation)                                                       | Status                                                                                                                                                                                                                                                                                                                                                                                                                              | Bedeutung                                                                                                                                                                                                                                                                                                        |
|-------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;table width=&gt;</code>                                       |                                                                                                 | Bestimmt die Breite der gesamten Tabelle in Bezug auf den verfügbaren Raum im Elternelement. Das Elternelement kann das »Dokument« sein, also das body-Element, oder auch ein anderes Element, innerhalb dessen eine Tabelle vorkommen kann, z.B. ein div-Element oder eine Tabellenzelle einer äußeren Tabelle. |
| <code>&lt;table height=&gt;</code>                                      |                                                                                                                                                                                                                                                                   | Bestimmt die Höhe der gesamten Tabelle in Bezug auf das Anzeigefenster. Diese Angabe wird zwar von den Browsern schon lange unterstützt und in der Praxis häufig benutzt, gehörte jedoch nie zum HTML-Standard.                                                                                                  |
| <code>&lt;th width=&gt;</code> oder<br><code>&lt;td width=&gt;</code>   |           | Bestimmt die Breite einer Tabellenzelle und damit die Breite aller Zellen der gleichen Spalte. Als <i>deprecated</i> eingestuft. Ersetzbar durch CSS-Notationen wie <code>&lt;th style="width:100px"&gt;</code> . Solche Angaben werden allerdings nur von neueren Browsern interpretiert.                       |
| <code>&lt;th height=&gt;</code> oder<br><code>&lt;td height=&gt;</code> |      | Bestimmt die Höhe einer Tabellenzelle und damit die Höhe aller Zellen der gleichen Tabellenzeile. Als <i>deprecated</i> eingestuft. Ersetzbar durch CSS-Notationen wie <code>&lt;td style="height:100px"&gt;</code> . Solche Angaben werden allerdings nur von neueren Browsern interpretiert.                   |

Einige Browser interpretieren darüber hinaus weitere Angaben wie `<tr height=>`. Solche Angaben sind allesamt nicht HTML-konform und sollten nicht mehr verwendet werden.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Breit und hoch mit HTML</h1>

```

```

<table border="1" width="60%">
  <tr>
    <td width="50%" height="100">Berlin</td>
    <td>Hamburg</td>
    <td>M&uuml;nchen</td>
  </tr><tr>
    <td>Milj&ouml;h</td>
    <td>Kiez</td>
    <td>Bierdampf</td>
  </tr><tr>
    <td>Buletten</td>
    <td>Frikadellen</td>
    <td>Fleischpflanzerl</td>
  </tr>
</table>

<h1>Breit und hoch mit CSS</h1>

<table border="1" style="width:60%">
  <tr>
    <td style="width:50%; height:100px">Berlin</td>
    <td>Hamburg</td>
    <td>M&uuml;nchen</td>
  </tr><tr>
    <td>Milj&ouml;h</td>
    <td>Kiez</td>
    <td>Bierdampf</td>
  </tr><tr>
    <td>Buletten</td>
    <td>Frikadellen</td>
    <td>Fleischpflanzerl</td>
  </tr>
</table>

</body>
</html>

```

Im Beispiel wird zweimal die gleiche Tabelle notiert – im ersten Fall kommen die HTML-Attribute `width=` und `height=` an erlaubten Stellen zum Einsatz – nach der HTML-Variante »Transitional«. Die zweite Tabelle benutzt entsprechende CSS-Eigenschaften zur Angabe von Breite und Höhe und genügt damit auch den Anforderungen an die HTML-Variante »Strict«. Interpretiert wird die zweite Tabelle allerdings erst vom Internet Explorer ab Version 4.x und von Netscape ab Version 6.x.

**Beachten Sie:** Die Angaben zu Breite und Höhe sind nur wirksam, wenn der Tabellen- bzw. Zelleninhalt kleiner ist, sodass leerer Raum entsteht. Wenn der Tabelleninhalt größer ist, werden die Angaben zu Breite und Höhe außer Kraft gesetzt. Die Tabelle bzw. Zelle (und mit ihr Spalte und Zeile) werden dann so dimensioniert, dass der gesamte Inhalt angezeigt wird. Einige

Browser – z.B. Netscape bis einschließlich Version 4.x – setzen Angaben zu Breite und Höhe allerdings leider je nach Tabelleninhalt stärker außer Kraft als es nötig wäre.

Da die Angabe zu `width=` in einer Tabellenzelle spaltenweit gilt, brauchen Sie sie nur einmal pro Spalte zu notieren. Am sinnvollsten ist es, die Angabe in der ersten Zeile der Tabelle zu notieren. Ebenso ist es mit der Angabe zu `height=` in einer Tabellenzelle. Da diese Angabe für die ganze zugehörige Tabellenzeile gilt, ist es am sinnvollsten, sie in der ersten Zeile der Zeile zu notieren. Im obigen Beispiel wird durch `<td width="50%" height="100">` in der ersten Zeile also sowohl die Breite der ersten Spalte als auch die Höhe der ersten Zeile definiert.

Verwenden Sie Pixelangaben und Prozentangaben immer so, dass keine Konflikte entstehen. Im obigen Beispiel werden etwa alle Breiten prozentual bestimmt, und nur die Höhe wird mit Pixeln festgelegt. So kann der Browser im Beispiel für die gesamte Tabelle eine Breite von 60% Prozent zum Elternelement ermitteln und für die erste Spalte wiederum 50% davon. Die Angabe von 100 Pixeln Zeilenhöhe kommt damit nicht in Konflikt.

Eine übersichtlichere Lösung, die Breite von Tabellenspalten zu definieren, steht mit der Möglichkeit zur Verfügung, Spalten vorzudefinieren (Kapitel 8.1.2). Die hier beschriebene Möglichkeit hat dagegen den Vorteil, auch von älteren Browsern interpretiert zu werden.

## 8.2.5 Zeilenumbruch in Zellen verhindern

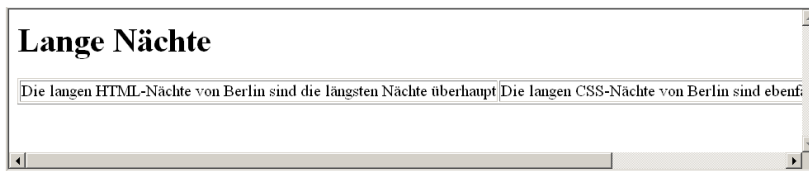
Sie können festlegen, dass beim Inhalt einer Zelle kein automatischer Zeilenumbruch erfolgen darf. Die Spalte der Tabelle wird bei der Anzeige entsprechend breit dimensioniert. Das entsprechende Attribut ist jedoch als *deprecated* eingestuft und soll künftig aus dem HTML-Standard entfallen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Lange N&auml;chte</h1>

<table border="1">
  <tr>
    <td nowrap>Die langen HTML-N&auml;chte von Berlin sind die l&auml;ngsten
N&auml;chte &uuml;berhaupt</td>
    <td style="white-space:nowrap">Die langen CSS-N&auml;chte von Berlin sind
ebenfalls die l&auml;ngsten N&auml;chte &uuml;berhaupt</td>
  </tr>
</table>

</body>
</html>
```



8.10: Ohne Umbruch lassen sich breite Tabellenzellen nur mit dem Rollbalken überblicken.

Mit dem Attribut `nowrap` (ohne weitere Wertzuweisung) in einem einleitenden `<th>`- oder `<th>`-Tag verhindern Sie, dass der Text innerhalb der Zelle automatisch umbrochen wird. Die Zelle und damit ihre gesamte zugehörige Spalte wird so breit wie erforderlich.

Das obige Beispiel zeigt in der ersten Zelle, wie das HTML-Attribut `nowrap` eingesetzt wird, und in der zweiten Zelle die entsprechende Notation in CSS. Letztere ist konform zur HTML-Variante »Strict«, wird jedoch erst ab Internet Explorer 4.x und Netscape 6.x interpretiert.

**Beachten Sie:** Wenn Sie **XHTML-konform** arbeiten, müssen Sie das `nowrap`-Attribut in der Form `nowrap="nowrap"` notieren. Weitere Informationen dazu im Kapitel 14.1.11 zu *XHTML und HTML*.

## 8.2.6 Ausrichtung von Zellen

Zelleninhalte von Tabellen können aufgrund unterschiedlicher Inhalte oder durch Angaben zu Breite und Höhe mehr Raum in Anspruch nehmen, als es ihr Inhalt erfordert. Für diesen Fall können Sie Zelleninhalte sowohl horizontal (linksbündig, zentriert, rechtsbündig) als auch vertikal (obenbündig, mittig, untenbündig) am Zellenrand ausrichten. Die entsprechenden Angaben sind sowohl auf einzelne Zellen (also `th`- und `td`-Elemente) anwendbar als auch auf Tabellenbereiche wie Tabellenzeilen (`tr`), Tabellenkopf (`thead`), Tabellenkörper (`tbody`) und Tabellenfuß (`tfoot`). In diesem Fall gelten die Angaben für alle Zellen des jeweiligen Bereichs. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Richtungsweisend</h1>

<table border="1">
  <tr>
    <th width="200" height="100" align="left" valign="top"> links oben</th>
    <th width="200">mittig zentriert</th>
    <th width="200" align="right" valign="bottom"> rechts unten</th>
  </tr><tr align="center" valign="top">
```



```

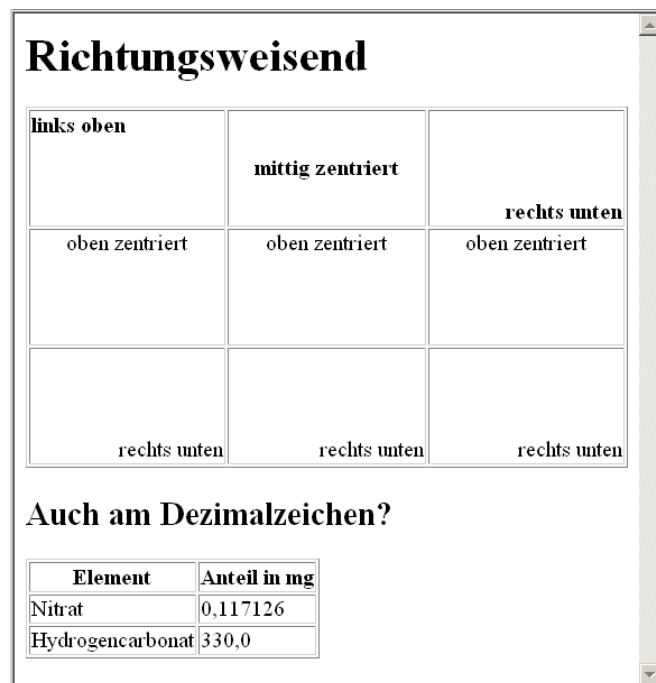
<td height="100">oben zentriert</td>
<td>oben zentriert</td>
<td>oben zentriert</td>
</tr><tr align="right" valign="bottom">
<td height="100">rechts unten</td>
<td>rechts unten</td>
<td>rechts unten</td>
</tr>
</table>

<h2>Auch am Dezimalzeichen?</h2>

<table border="1">
<colgroup><col><col align="char" char=","></colgroup>
<tr><th>Element</th><th>Anteil in mg</th></tr>
<tr><td>Nitrat</td><td>0,117126</td></tr>
<tr><td>Hydrogencarbonat</td><td>330,0</td></tr>
</table>

</body>
</html>

```



8.11: Die Zellinhalte sind mit Attributen waagrecht und senkrecht ausgerichtet.

Mit `align="left"` können Sie eine Kopfzeile in ihrem einleitenden `<th>`-Tag linksbündig ausrichten, mit `align="right"` rechtsbündig. Datenzellen können Sie in ihrem einleitenden `<td>`-Tag mit `align="center"` zentriert ausrichten und durch `align="right"` rechtsbündig. Die Angaben `align="center"` bei Kopfzellen und `align="left"` bei Datenzellen sind natürlich auch erlaubt, bewirken aber lediglich die Voreinstellung und sind deshalb nicht nötig (*align = Ausrichtung, left = links, center = zentriert, right = rechts*). Diese Angaben können Sie auch in einleitenden `<tr>`-Tags, einleitenden `<thead>`-Tags, einleitenden `<tbody>`-Tags und `<tfoot>`-Tags notieren. Dann werden alle Zellen im entsprechenden Bereich so ausgerichtet wie angegeben. Widersprechende Angaben in einzelnen Zellen haben allerdings Vorrang vor bereichsweiten Angaben. Ab HTML 4.0 ist auch die Angabe `align="justify"` erlaubt. Dadurch erzwingen Sie einen Blocksatz innerhalb der Zelle.

Mit `valign="top"` können Sie eine Zelle in ihrem einleitenden Tag obenbündig ausrichten, durch die Angabe `valign="bottom"` untenbündig. Auch die Angabe `valign="middle"` ist möglich. Dies ist jedoch die Voreinstellung und deshalb nicht unbedingt nötig (*valign = vertikale Ausrichtung, top = oben, bottom = unten, middle = mittig*). Auch das `valign`-Attribut können Sie in einleitenden `<tr>`-Tags, einleitenden `<thead>`-Tags, einleitenden `<tbody>`-Tags und `<tfoot>`-Tags notieren. Ab HTML 4.0 ist auch die Angabe `valign="baseline"` erlaubt. Dann werden alle Zellen innerhalb einer Zeile, die diese Angaben erhalten, an einer gemeinsamen Basislinie ausgerichtet. Und zwar so, dass die erste Textzeile aller dieser Zellen immer auf gleicher Höhe beginnt.

Ferner ist seit HTML 4.0 die Angabe `align="char"` in Verbindung mit den Attributen `char=` und `charoff=` erlaubt. Mit `align="char"` können Sie erzwingen, dass die Spalteninhalte an einem Dezimalzeichen ausgerichtet werden. Außerdem benötigen Sie noch das Attribut `char=` (*char = character = Zeichen*). Diesem weisen Sie als Wert ein Zeichen zu, das Sie als Dezimalzeichen verwenden. Die Ausrichtung erfolgt dann so, dass das Dezimalzeichen immer an der gleichen Stelle steht. Neben dem Attribut `char=` zur Bestimmung des Dezimalzeichens können Sie zusätzlich noch das Attribut `charoff=` angeben (*charoff = character offset = Zeichenposition*). Damit können Sie angeben, an welcher Position das Zeichen frühestens vorkommen kann. Im obigen Beispiel wird ein Komma als Dezimalzeichen definiert. Die Werte der betreffenden im Beispiel vordefinierten Spalten – werden dann so ausgerichtet, dass das Dezimalzeichen immer an der gleichen Stelle steht. Die HTML-Spezifikation gibt jedoch an, dass die Unterstützung der Dezimalzeichenausrichtung durch Browser freiwillig ist. Und tatsächlich interpretiert auch keiner der weit verbreiteten Browser diese Angaben.

### 8.2.7 Hintergrundfarben und Hintergrundbilder

Sie können sowohl für eine gesamte Tabelle als auch für einzelne Zeilen oder Zellen eine Hintergrundfarbe definieren. Die verbreiteten Web-Browser unterstützen auch die Angabe von Hintergrundbildern mit Wallpaper-Effekt, doch diese Angaben gehörten nie zum HTML-Standard. Auch die Angaben zur Hintergrundfarbe sind als *deprecated* gekennzeichnet und sollen künftig aus dem HTML-Standard entfallen. Sowohl das Definieren von Hintergrundfarben als auch das von Hintergrundbildern ist mit CSS Stylesheets möglich.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body text="#000099">

<h1>Großzügige Worte mit HTML</h1>

<table border="1" bgcolor="#FFFF00">
<tr>
  <td width="200" height="100"><h2>Traum</h2></td>
  <td width="200" bgcolor="#00FFFF"><h2>Zeit</h2></td>
  <td width="200"><h2>Luft</h2></td>
</tr><tr bgcolor="#FF00FF">
  <td height="100"><h2>Licht</h2></td>
  <td><h2>Himmel</h2></td>
  <td><h2>Leben</h2></td>
</tr>
</table>

<h1>Großzügige Worte mit CSS</h1>

<table border="1" style="background-color:#FFFF00">
<tr>
  <td width="200" height="100"><h2>Traum</h2></td>
  <td width="200" style="background-color:#00FFFF"><h2>Zeit</h2></td>
  <td width="200"><h2>Luft</h2></td>
</tr><tr style="background-color:#FF00FF">
  <td height="100"><h2>Licht</h2></td>
  <td><h2>Himmel</h2></td>
  <td style="background-image:url(background.jpg)"><h2>Leben</h2></td>
</tr>
</table>

</body>
</html>
```

Durch das Attribut `bgcolor=` im einleitenden Tag der Tabelle können Sie eine Hintergrundfarbe für die gesamte Tabelle bestimmen (*bgcolor* = *background color* = *Hintergrundfarbe*). Beim Angeben der Farbe gelten die Regeln zum Definieren von Farben in HTML. Ebenso können Sie das Attribut `bgcolor=` im einleitenden Tag einer Tabellenzeile (`<tr>`) notieren, um die Hintergrundfarbe für alle Zellen in dieser Zeile zu bestimmen. Wenn Sie `bgcolor=` im einleitenden Tag einer Datenzeile (`<td>`) oder einer Kopfzeile (`<th>`) angeben, gilt die Hinter-

grundfarbe für diese eine Zelle. Im Konfliktfall hat die Farbangabe in einzelnen Zellen Vorrang vor der Angabe für ganze Zeilen oder Tabellen. Die Angabe für eine Zeile hat im Konfliktfall Vorrang vor der Angabe für die ganze Tabelle.



8.12: Hintergrundfarben und -bilder für Tabellen mit Attributen und CSS-Angaben.

Das Gleiche ist mit dem Attribut `background=` möglich, um eine Hintergrundgrafik für die Tabelle oder einzelne Zellen einzubinden. Dieses Attribut ist jedoch im Zusammenhang mit Tabellen nicht HTML-konform und kommt im obigen Beispiel auch nicht vor. Die zweite im Beispiel notierte Tabelle zeigt, wie es HTML-konform gemacht wird. Dagegen spricht eigentlich nur noch die Existenz von Netscape 4.x, der keine CSS-Angaben bei Tabellen interpretiert.

Im obigen Beispiel wird angenommen, dass sich die Grafikdatei `background.jpg` im gleichen Verzeichnis befindet. Sie können jedoch auch Grafiken aus anderen Verzeichnissen (z.B. `/daten/grafik/hintergrund/background.gif` oder `../gif/back.gif`) oder von entfernten URIs (z.B. `http://www.mein-anderer-server.de/grafik/back.jpg`) angeben.

**Beachten Sie:** Wenn Sie eine Hintergrundfarbe definieren oder eine Hintergrundgrafik einbinden, sollten Sie für den Text eine geeignete Kontrastfarbe definieren.

### 8.2.8 Farben für Rand und Gitternetzlinien (Microsoft)

Sie können auch für Rahmen und Gitternetzlinien Farbangaben machen. Diese Angaben werden bislang jedoch nur vom MS Internet Explorer interpretiert und gehören nicht zum HTML-Standard. Benutzen Sie stattdessen CSS zur Gestaltung der Rahmen.

Durch das Attribut `bordercolor=` im einleitenden `<table>`-Tag der Tabelle können Sie eine einheitliche Farbe für den Tabellenrahmen und die Gitternetzlinien bestimmen (*bordercolor = Randfarbe*). Anstelle der einfachen Rahmenfarbe können Sie auch einen Schattier-Effekt in den Rahmen bringen, indem Sie zwei verschiedene Farben definieren – eine dunklere und eine hellere. Dazu notieren Sie im einleitenden Tag der Tabelle die beiden Attribute `bordercolor-dark=` (*bordercolordark = dunkle Randfarbe*) und `bordercolor-light=` (*bordercolorlight = helle Randfarbe*).

## 8.3 Tabellen mit CSS gestalten

Gerade bei Tabellen sind CSS Stylesheets eine große Hilfe zur Gestaltung. Denn mit CSS können Sie alle Elemente einer Tabelle nach Wunsch formatieren – sei es einzeln oder gruppenweise. Dazu kommen ausgereifte CSS-Eigenschaften für Tabellenrahmen. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: Tabellenformatierung, Schriftformatierung, Ausrichtung und Absatzkontrolle, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder, Positionierung und Anzeige von Elementen.

## 8.4 Zellen verbinden

### 8.4.1 Zellen in einer Zeile spaltenweise verbinden

Sie können mehrere Zellen innerhalb einer Zeile verbinden, sodass sich eine Spalte in dieser Zeile über mehrere Spalten hinweg erstreckt. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Zoologie</h1>

<table border="1">
  <tr>
    <th colspan="2">Die Menschheit besteht aus</th>
  </tr><tr>
    <td>Eseln</td>
```

```

    <td>Affen</td>
  </tr>
</table>

</body>
</html>

```



8.13: In der ersten Zeile dieser Tabelle sind zwei Zellen zusammengefasst.

Durch das Attribut `colspan= [Anzahl Spalten]` erreichen Sie, dass sich eine Zelle über mehrere Spalten hinweg erstreckt (*colspan = column span = Spalten überspannen*). Die Angabe ist nur wirksam, wenn die Tabelle mindestens so viele Spalten besitzt wie angegeben.

**Beachten Sie:** Wenn Sie in einer Tabellenzeile eine Zelle beispielsweise über vier Spalten hinweg »spannen«, sollten Sie in den anderen Zeilen der Tabelle anstelle dieser einen Zelle vier eigene Zellen definieren. Die Summe der Zellen sollte in jeder Zeile der Tabelle gleich sein.

## 8.4.2 Zellen in einer Spalte zeilenweise verbinden

Sie können mehrere Zellen in einer Spalte verbinden, sodass sich eine Zeile in dieser Spalte über mehrere Zeilen hinweg erstreckt. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Zoologie (II)</h1>

<table border="1">
  <tr>
    <th rowspan="2">Die Eselheit besteht aus</th>
    <td>echten Eseln</td>
  </tr><tr>
    <td>verkappten Eseln (Menschen)</td>
  </tr><tr>
    <th rowspan="2">Die Affenheit besteht aus</th>

```

```

<td>echten Affen</td>
</tr><tr>
  <td>verkappten Affen (Menschen)</td>
</tr>
</table>

</body>
</html>

```

|                           |                             |
|---------------------------|-----------------------------|
| Die Eselheit besteht aus  | echten Eseln                |
|                           | verkappten Eseln (Menschen) |
| Die Affenheit besteht aus | echten Affen                |
|                           | verkappten Affen (Menschen) |

8.14: Dieses Beispiel fasst in der ersten Spalte jeweils zwei untereinander liegende Zellen zusammen.

Durch das Attribut `rowspan= [Anzahl Zeilen]` erreichen Sie, dass sich eine Zelle in einer Spalte über mehrere Zeilen hinweg erstreckt (*rowspan = Zeilen spannen*). Die Angabe ist nur wirksam, wenn die Tabelle mindestens so viele Zeilen besitzt wie angegeben.

**Beachten Sie:** Wenn Sie in einer Tabellenzeile eine Zelle beispielsweise über vier Zeilen hinweg »spannen«, sollten Sie in der gleichen Zeile den Inhalt der ersten daneben stehenden Zelle definieren. In den drei folgenden Zeilen sollten Sie an gleicher Stelle die drei übrigen Zellen definieren, die neben der gestreckten Zelle stehen sollen.

### 8.4.3 Zellen zugleich spalten- und zeilenweise verbinden

Sie können Zellen definieren, die sich gleichzeitig über mehrere Zeilen und Spalten hinweg erstrecken. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Zoologie (III)</h1>

<table border="1">

```

```

<tr>
  <th colspan="2" rowspan="2">Der Mensch ist...</th>
  <td>zum einen Teil ein Esel</td>
</tr><tr>
  <td>zum anderen Teil ein Affe</td>
</tr><tr>
  <td>weder ein richtiger Esel</td>
  <td>noch ein richtiger Affe</td>
  <td><b>menschlich, allzumenschlich</b></td>
</tr>
</table>

</body>
</html>

```

| Der Mensch ist...        |                         | zum einen Teil ein Esel            |
|--------------------------|-------------------------|------------------------------------|
|                          |                         | zum anderen Teil ein Affe          |
| weder ein richtiger Esel | noch ein richtiger Affe | <b>menschlich, allzumenschlich</b> |

8.15: Das Beispiel fasst gleichzeitig Zellen waagrecht und senkrecht zusammen.

Sie können die Attribute `colspan= [Anzahl Spalten]` und `rowspan= [Anzahl Zeilen]` in einer Zelle kombinieren. Dadurch erreichen Sie, dass sich die Zelle über mehrere Spalten und Zeilen zugleich erstreckt.

## 8.5 Tabellenbeschriftung und Tabellenausrichtung

### 8.5.1 Tabellenüberschrift/-unterschrift definieren

Sie können für eine Tabelle eine von der Tabelle getrennte, aber ihr zugehörige Über- oder Unterschrift definieren. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titles</title>
</head>
<body>

```



```

<table border="1">
  <caption><big>Assoziationen</big></caption>
  <tr>
    <th>Berlin</th>
    <th>Hamburg</th>
    <th>M&uuml;nchen</th>
  </tr><tr>
    <td>Milj&ouml;h</td>
    <td>Kiez</td>
    <td>Bierdampf</td>
  </tr><tr>
    <td>Buletten</td>
    <td>Frikadellen</td>
    <td>Fleischpflanzerl</td>
  </tr>
</table>

</body>
</html>

```



| Berlin   | Hamburg     | München          |
|----------|-------------|------------------|
| Miljöh   | Kiez        | Bierdampf        |
| Buletten | Frikadellen | Fleischpflanzerl |

8.16: Eine Tabelle mit Überschrift.

`<caption>...</caption>` definiert eine Tabellenüberschrift. Das Element muss unmittelbar nach dem einleitenden `<table>`-Tag notiert werden. Es bewirkt normalen Fließtext. Um den enthaltenen Text auffällig zu gestalten, darf das `caption`-Element Inline-Elemente als weitere Elemente enthalten. Eine andere Möglichkeit besteht darin, das `caption`-Element mit CSS Stylesheets zu formatieren.

Wenn Sie nichts anderes angeben, bewirkt das `caption`-Element eine Tabellenüberschrift. Um eine Tabellenunterschrift zu erzwingen, können Sie `<caption align="bottom">` notieren. Ferner sind die Angaben `align="left"` für seitliche Überschrift links und `align="right"` für seitliche Überschrift rechts erlaubt. Während `align="bottom"` von den Browsern interpretiert wird, werden `align="left"` und `align="right"` meistens nicht oder nicht korrekt unterstützt. Das Attribut `align=` ist darüberhinaus als *deprecated* gekennzeichnet, soll also künftig aus dem HTML-Standard entfallen. Alternativangaben in CSS werden von den weit verbreiteten Browsern aber im Zusammenhang mit dem `caption`-Element bislang nicht unterstützt.

## 8.5.2 Tabellenausrichtung

Sie können eine Tabelle links, zentriert oder rechts ausrichten. Die bekannten Browser reagieren dabei so, dass eine links oder rechts ausgerichtete Tabelle, die nicht die gesamte zur Verfügung stehende Breite beansprucht, bewirkt, dass nachfolgende Inhalte um die Tabelle herumfließen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titles</title>
</head>
<body>

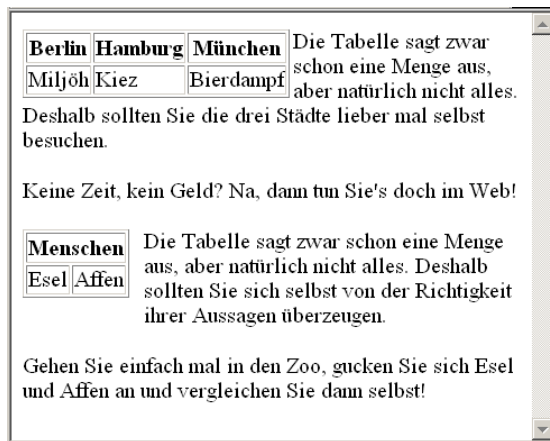
<table border="1" align="left">
  <tr>
    <th>Berlin</th>
    <th>Hamburg</th>
    <th>M&uuml;nchen</th>
  </tr><tr>
    <td>Milj&ouml;h</td>
    <td>Kiez</td>
    <td>Bierdampf</td>
  </tr>
</table>

<p>Die Tabelle sagt zwar schon eine Menge aus, aber nat&uuml;rlich nicht alles.
Deshalb sollten Sie die drei St&auml;dte lieber mal selbst besuchen.<br
clear="all"></p>
<p>Keine Zeit, kein Geld? Na, dann tun Sie's doch im Web!</p>

<table border="1" style="float:left; margin-right:10px; margin-bottom:6px">
  <tr>
    <th colspan="2">Menschen</th>
  </tr><tr>
    <td>Esel</td>
    <td>Affen</td>
  </tr>
</table>

<p>Die Tabelle sagt zwar schon eine Menge aus, aber nat&uuml;rlich nicht alles.
Deshalb sollten Sie sich selbst von der Richtigkeit ihrer Aussagen
&uuml;berzeugen.</p>
<p style="clear:left">Gehen Sie einfach mal in den Zoo, gucken Sie sich Esel und
Affen an und vergleichen Sie dann selbst!</p>

</body>
</html>
```



8.17: Die Position von Tabellen im Fliesstext regeln Sie mit Attributen oder CSS.

Im Beispiel sind zwei Tabellen und unterhalb davon jeweils zwei Textabsätze notiert. Die obere Tabelle ist mit `align="left"` im einleitenden `<table>`-Tag linksbündig ausgerichtet. Bei Netscape und beim Internet Explorer hat das die Wirkung, dass nachfolgender Text rechts um die Tabelle fließt. Mit `align="right"` können Sie eine Tabelle rechtsbündig ausrichten und mit `align="center"` zentriert. Das `align`-Attribut ist jedoch als *deprecated* gekennzeichnet und soll künftig aus dem HTML-Standard entfallen. Die zweite Tabelle im obigen Beispiel zeigt daher, wie sich der gleiche Effekt mit CSS-Formaten erzielen lässt.

Bei der Variante mit den CSS-Formaten sind außerdem noch Angaben zum Abstand rechts und unten angegeben, sodass der nachfolgende Text nicht so sehr an der Tabelle »klebt«. Die Browser interpretieren auch vergleichbare HTML-Attribute dafür: Mit `<table align="center" hspace="10" vspace="6">` erreichen Sie beispielsweise, dass zwischen Tabelle und umfließendem Text ein Horizontalabstand von 10 Pixeln und ein Vertikalabstand von 6 Pixeln erzwungen wird. Die Attribute `hspace=` und `vspace=` im Zusammenhang mit Tabellen gehörten jedoch nie zum HTML-Standard.

Die beiden obigen Beispiele zeigen außerdem, wie Sie den Textumfluss abbrechen und die Textfortsetzung unterhalb der Tabelle erzwingen können. In der ersten Tabelle des Beispiels, wo alles mit HTML realisiert ist, ist dazu das Tag `<br clear="all">` notiert – also ein Zeilenumbruch mit einem Attribut, das für die Fortsetzung unterhalb des Umflussbereichs sorgt. Das `clear`-Attribut in HTML ist jedoch ebenfalls als *deprecated* gekennzeichnet. Der Text unterhalb der zweiten Tabelle im Beispiel zeigt, wie der gleiche Effekt mit CSS-Formaten erreichbar ist.

**Beachten Sie:** Wenn die Tabelle selbst größere Inhalte hat, müssen Sie eine Gesamtbreite der Tabelle erzwingen (`<table width=`, Kapitel 8.2.4), damit links bzw. rechts noch Platz bleibt für den umfließenden Text. Wenn Sie eine Tabelle zwar linksbündig oder rechtsbündig ausrichten, aber zugleich verhindern wollen, dass nachfolgende Inhalte um die Tabelle fließen, dann schließen Sie die gesamte Tabelle in ein allgemeines Blockelement (`div`) ein und richten dieses mit `align=` oder einer entsprechenden CSS-Eigenschaft (`style="text-align:"`) aus.

### 8.5.3 Weitere Informationen

Das `caption`-Element kann auch Universalattribute enthalten.

## 8.6 Tabellen als Mittel für Web-Seitenlayouts

### 8.6.1 Allgemeines zu Tabellen für Web-Seitenlayouts

Wenn Sie bei Web-Seiten auf sauber ausgerichtete, aber relativ frei verteilte Elemente mit meist farbigen Flächen stoßen, wurde in vielen Fällen mit der Technik der »blinden Tabellen« gearbeitet. Blinde Tabellen haben keine sichtbaren Gitternetzlinien. Dadurch merkt der Betrachter gar nicht, dass es sich in Wirklichkeit um eine Tabelle handelt.

Einen Nachteil sollten Sie jedoch beachten, wenn Sie beabsichtigen, den gesamten anzuzeigenden Inhalt einer Web-Seite mit Hilfe einer blinden Tabelle anzuordnen: Tabellen werden vom Web-Browser erst angezeigt, nachdem ihr kompletter Inhalt eingelesen wurde bzw. erst in dem Augenblick, wo der Web-Browser genau weiß, wie groß und wie breit die Tabelle angezeigt werden muss. Das bedeutet bei der Datenübertragung aus dem Web, dass ein Anwender am Bildschirm länger wartet, bis überhaupt etwas angezeigt wird – es sei denn, Sie benutzen die Möglichkeit, Spalten vorzudefinieren (Kapitel 8.1.2).

Ein weiteres Problem besteht darin, dass Browser mit gewünschten Breiten- und Höhenangaben nicht unbedingt so umgehen, wie Sie es wünschen. Inhalte werden dann plötzlich gestaucht, auseinandergezogen usw. Dies hat zur Erfindung des so genannten »blinden Pixels« geführt, einer kleinen GIF-Grafik, die nur aus einer einfarbigen Fläche besteht, deren Farbe wiederum als transparent definiert ist. Zusammen mit der Möglichkeit, Breite und Höhe einer Grafik in HTML (Kapitel 7.1.3) unabhängig von der tatsächlichen Größe der Grafik anzugeben, lässt sich eine solche Grafik scheinbar unsichtbar und unauffällig einbauen. So lassen sich vor allem feste Mindestbreiten für einzelne Spalten einer Tabelle erzwingen, und der Browser kann die Inhalte darin nicht mehr stauchen. Diese Technik gilt einerseits als unsaubere Trickserie, ist aber andererseits Ausdruck für fehlende Formatierungsmöglichkeiten. Mittlerweile stellen CSS Stylesheets solche Möglichkeiten zwar bereit (vgl. die CSS-Eigenschaft `min-width`), doch das nutzt nicht viel, solange die weit verbreiteten Browser noch keine Unterstützung dafür anbieten.

In diesem Abschnitt wird anhand eines Beispiels gezeigt, wie Seitenlayouts mit blinden Tabellen im Prinzip funktionieren. Auf die Technik des blinden Pixels wird hier verzichtet. Doch eine andere, typische Technik kommt zum Einsatz: das Verschachteln von Tabellen. Denn meistens lassen sich saubere Tabellenlayouts nur erreichen, wenn man mit Tabellen innerhalb von Tabellen arbeitet.

### 8.6.2 Beispiel eines typischen Tabellen-Layouts

Das folgende Beispiel zeigt ein Layout, wie man es auf vielen Web-Seiten findet: oben eine farbige Fläche, wo die Seitenüberschrift steht und andere Elemente wie Grafiken für Logos, Banner oder dergleichen Platz haben, unterhalb davon links eine gleichfarbige, direkt angeschlossene, schmale Fläche für Navigationsverweise innerhalb des Web-Projekts, und rechts davon eine breite Fläche für den eigentlichen Inhalt der Seite.



8.18: So sieht das nachfolgende Beispiel-Listing im Internet Explorer aus.

Das Beispiel ist gültiges HTML 4.0, benutzt aber diverse Attribute, die als *deprecated* gekennzeichnet sind, weshalb als HTML-Variante »Transitional« gewählt werden muss.

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#FFFFFF" link="#FFFF00" vlink="#FFCC00" alink="#FFFFFF"
text="#000000">

<table border="0" cellpadding="0" cellspacing="0" width="100%" bgcolor="#A050A0">
<colgroup>
  <col width="200">
    <col>
  </colgroup>
<tr>
  <td colspan="2">
    <table border="0" cellpadding="10" cellspacing="0">
```

```

<tr>
  <td>
    <h1 style="color:#FFFFFF">Willkommen</h1>
  </td>
</tr>
</table>
</td>
</tr><tr>
<td valign="top" width="200">
  <table border="0" cellpadding="10" cellspacing="0">
    <tr>
      <td>
        <h3 style="color:#FFFFFF">Navigation</h3>
        <p><a href="table_tr_th_td.htm"><b>Tabelle</b></a><br>
        <a href="bgcolor.htm"><b>Hintergrund</b></a><br>
        <a href="colspan.htm"><b>Zellen verbinden</b></a><br>
        <a href="width_height.htm">
          <b>Breite und H&ouml;he</b>
        </a></p>
        <h3 style="color:#FFFFFF">News</h3>
        <p style="color:#FFFFFF">Die Neuigkeiten stehen
          stets an dieser Stelle.
          Viel Neues gibt es nicht. Nur so viel: diese
          Seite ist neu.
        </p>
      </td>
    </tr>
  </table>
</td>
<td valign="top" bgcolor="#FFFFFF">
  <table border="0" cellpadding="10" cellspacing="0">
    <tr>
      <td>
        <h2>Worum geht es hier eigentlich?</h2>
        <p>Ach, um Web-Design, halt das &Uuml;bliche.
          viel Inhalt viel Inhalt viel Inhalt
          viel Inhalt viel Inhalt viel Inhalt viel Inhalt.
        </p>
        <p>viel Inhalt</p><p>viel Inhalt</p>
        <p>viel Inhalt</p><p>viel Inhalt</p>
        <p>viel Inhalt</p><p>viel Inhalt</p>
        <p>viel Inhalt</p><p>viel Inhalt</p>
      </td>
    </tr>
  </table>
</td>
</tr>

```

```

</table>

</body>
</html>

```

Das Beispiel bestimmt im einleitenden `<body>`-Tag zunächst dateiweite Farben für Hintergrund, Text und Verweise (Kapitel 4.2.1). Innerhalb des Dateikörpers wird die übergeordnete Tabelle definiert. Sie erhält die Attribute `border="0"`, damit kein Rahmen und keine Gitternetzlinien angezeigt werden, sowie `cellpadding="0"` und `cellspacing="0"`, damit sich die einzelnen Zellen der Tabelle nahtlos aneinander fügen. Das ist wichtig, damit zwischen den Farbflächen keine sichtbaren Lücken entstehen.

Weil es sich um eine die ganze Seite überspannende Tabelle handelt, werden für Browser, die das verstehen, die gewünschten Spalten mit `<col>`-Tags vordefiniert. Die Tabelle soll zwei Spalten haben, wobei die erste eine Breite von 200 Pixeln haben soll. Das soll die Spalte für Navigationsverweise werden. Die zweite Spalte soll den Rest des verfügbaren Platzes in Anspruch nehmen. Die gesamte Tabelle soll sich über die volle verfügbare Breite erstrecken – angegeben durch `width="100%"` im einleitenden `<table>`-Tag der übergeordneten Tabelle.

Die gesamte übergeordnete Tabelle bekommt ferner mit Hilfe von `bgcolor="#A050A0"` einen matt-violetten Hintergrund zugewiesen. Damit erhalten erst mal alle ihre Inhalte diese Hintergrundfarbe. Um andersfarbige Flächen innerhalb der Tabelle zu erzeugen, sind bei den einzelnen Zellen weitere Angaben zur Hintergrundfarbe erforderlich, die dann diese globale Hintergrundfarbe überschreiben.

Die erste Zelle der ersten Zeile der übergeordneten Tabelle ist zugleich die einzige Zelle in dieser Zeile, da durch das Attribut `colspan="2"` erreicht wird, dass sich die Zelle über zwei Spalten erstreckt. Der Inhalt dieser Zelle steht nun für den Überschriftenteil zur Verfügung. Da mit `cellpadding="0"` und `cellspacing="0"` gearbeitet wird, würde der Inhalt allerdings ganz am Außenrand der Tabelle kleben, was unsauber aussieht.

Deshalb kommt bereits bei der Überschrift zum ersten Mal die Technik der Tabelle in der Tabelle zum Einsatz: `<table border="0" cellpadding="10" cellspacing="0">` definiert innerhalb der über zwei Spalten gestreckten Tabellenzelle eine weitere Tabelle. Auch diese ist wiederum rahmenlos (`border="0"`). Durch `cellpadding="10"` wird jedoch ein Innenabstand von 10 Pixeln zum Zellenrand erzwungen. Diese Angabe ist der eigentliche Grund, warum die innere Tabelle definiert wird. Denn die innere Tabelle hat nur eine einzige Zeile mit einer einzigen Spalte, also nur eine Zelle, und darin ist die `h1`-Überschrift für die Seite notiert (»Willkommen«). Die Überschrift erhält mit Hilfe der CSS-Eigenschaft `color`: eine weiße Farbe, um sich ordentlich von dem violetten Hintergrund der Tabelle abzuheben.

In der zweiten Zeile der übergeordneten Tabelle werden für beide definierten Spalten je eine Zelle definiert. Die erste (linke) Zelle ist dann der Behälter für die gesamte linke Spalte – also für die Spalte, in der die Navigationsverweise stehen. Die zweite (rechte) Zelle ist der Behälter für den gesamten Inhalt des Hauptbereichs, also des Bereichs für den eigentlichen Seiteninhalt. Die zweite Zelle, also die für den Hauptbereich, erhält mit `bgcolor="#FFFFFF"` als Hintergrundfarbe weiß zugewiesen. So entsteht letztlich der optische Eindruck, als ob die Seite nur oben und links einen farbigen Balken enthalten würde, während der gesamte Rest der Seite einen weißen Hintergrund hat.

In beiden Zellen der zweiten Zeile der übergeordneten Tabelle wird auch wieder je eine innere Tabelle definiert, die mit den gleichen Attributen wie im Fall der Überschrift für saubere Abstände zum Rand sorgt. So werden auch alle Inhalte ordentlich auf einer Höhe ausgerichtet. Mit `valign="top"` wird in beiden Zellen dafür gesorgt, dass die Inhalte, die ja unterschiedlich groß sein können, auf jeden Fall obenbündig ausgerichtet werden.

Die Verweise in der linken Spalte heben sich durch ihre dateiweit definierten Farben gut von dem violetten Hintergrund ab. Normaler Text, der in der linken Spalte steht, also Überschriften, Textabsätze usw., erhalten im Beispiel wieder mit Hilfe von CSS eine weiße Textfarbe, um sich vom Hintergrund abzuheben.

**Beachten Sie:** Sie können CSS Stylesheets noch wesentlich konsequenter einsetzen, als es im obigen Beispiel der Fall ist. Maßgeblich sind im hier beschriebenen Zusammenhang z.B. folgende CSS-Eigenschaften: Schriftformatierung, Ausrichtung und Absatzkontrolle, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder, Positionierung und Anzeige von Elementen.

## 8.7 Tabellen und nichtvisuelle Medien

### 8.7.1 Allgemeines

Die folgenden Beschreibungen sind für die Ausgabe von Tabellen auf nichtvisuellen Medien gedacht. Vor allem bei Sprachausgabesystemen, die HTML-Dateien für Sehbehinderte ausgeben, stellen Tabellen normalerweise ein Problem dar. Denn die optische Eindeutigkeit einer Tabelle am Bildschirm oder auf Papier lässt sich in einem Medium, das Daten hintereinander ausgeben muss, nicht abbilden. Was beispielsweise in der optischen Abbildung einer Tabelle durch Gitternetzlinien ganz einfach ist, ist bei der Sprachausgabe ein echtes Problem: Wie soll akustisch verdeutlicht werden, wo eine Zelle beginnt und endet und zu welcher Spalte oder Zeile sie gehört? Das W3-Konsortium hat zu diesem Zweck eine Reihe von Attributen in HTML eingeführt, die speziell solche Probleme lösen helfen sollen.

### 8.7.2 Zusammenfassung des Tabelleninhalts

Sie können einen zusammenfassenden Kurztext für die Tabelle definieren. Ein Sprachausgabesystem könnte diesen Text als Einleitung zur Tabelle ausgeben. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<table border="1" summary="Die folgende Tabelle ordnet St&auml;nden typische
Begriffe zu.">
<tr>
```



```

<th>Berlin</th>
<th>Hamburg</th>
<th>M&uuml;nchen</th>
</tr><tr>
<td>Milj&ouml;h</td>
<td>Kiez</td>
<td>Bierdampf</td>
</tr><tr>
<td>Buletten</td>
<td>Frikadellen</td>
<td>Fleischpflanzerl</td>
</tr>
</table>

</body>
</html>

```

Mit dem Attribut `summary=` können Sie im einleitenden `<table>`-Tag eine Zusammenfassung der Tabelle definieren (*summary = Zusammenfassung*).

### 8.7.3 Kurzbeschreibung für Zellen definieren

Sie können für einzelne Tabellenzellen eine Kurzbeschreibung im einleitenden Tag der Zelle definieren. Ein Sprachausgabesystem könnte diesen Kurztext vor dem eigentlichen Inhalt ausgeben. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<table border="1">
  <tr>
    <th>Berlin</th>
    <th>Hamburg</th>
    <th>M&uuml;nchen</th>
  </tr>
  <tr>
    <td abbr="Es folgen Infos zu Berlin">
      Hier eine Menge Inhalt
    </td>
    <td abbr="Es folgen Infos zu Hamburg">
      Hier eine Menge Inhalt
    </td>

```

```

        <td abbr="Es folgen Infos zu München">
            Hier eine Menge Inhalt
        </td>
    </tr>
</table>

</body>
</html>

```

Mit dem Attribut `abbr=` im einleitenden Tag einer Datenzelle (`<td>`) oder einer Kopfzelle (`<th>`) können Sie eine Kurzinformation zu der entsprechenden Zelle voranschicken (*abbr* = *abbreviated* = *verkürzt*).

#### 8.7.4 Kopfzellen in Zellen referenzieren

Für nichtvisuelle Ausgabemedien, etwa für Sprachausgabesysteme, können Sie bestimmen, dass bei der Ausgabe einer Datenzelle ein expliziter Bezug zu einer oder mehreren Kopfzellen hergestellt wird. Das System könnte den Inhalt der Kopfzelle(n) dann bei der Datenzelle wiederholen. Ansonsten ist es bei der Sprachausgabe sehr schwer, den Bezug zwischen einer Datenzelle und den zugehörigen Kopfzellen herzustellen, da die optische Eindeutigkeit einer Tabelle dort nicht vorhanden ist. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<table border="1">
    <tr>
        <th id="Stadt_1">Berlin</th>
        <th id="Stadt_2">Hamburg</th>
        <th id="Stadt_3">München</th>
    </tr>
    <tr>
        <td headers="Stadt_1">Buletten</td>
        <td headers="Stadt_2">Frikadellen</td>
        <td headers="Stadt_3">Fleischpflanzerl</td>
    </tr>
</table>

</body>
</html>

```

Um Bezüge zwischen Datenzellen und Kopfzellen nach dem obigen Beispiel herzustellen, müssen Sie für die Kopfzellen mit dem Universalattribut `id=` individuelle Namen vergeben (im

obigen Beispiel Stadt\_1, Stadt\_2 und Stadt\_3). Innerhalb einer Datenzelle können Sie dann mit dem Attribut `headers=` auf Kopfzellen Bezug nehmen (*headers = Kopfzellen*). Im obigen Beispiel könnte ein Sprachausgabesystem durch den Bezug also etwa ausgeben: »Berlin ist gleich Bulletten«.

Falls Sie eine Kreuztabelle haben, bei der zwei Kopfzellen zu einer Datenzelle gehören, können Sie hinter `headers=` auch weitere id-Namen von Kopfzellen angeben. Trennen Sie die id-Namen durch Leerzeichen. Die Angabe hinter `headers=` sollte in jedem Fall in Anführungszeichen stehen.

Der umgekehrte Weg – nämlich das Wiederholen der Kopfzelleninformation in jeder zugehörigen Datenzelle – ist ebenfalls möglich. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<table border="1">
  <tr>
    <th scope="col">Berlin</th>
    <th scope="col">Hamburg</th>
    <th scope="col">M&uuml;nchen</th>
  </tr>
  <tr>
    <td>Bulletten</td>
    <td>Frikadellen</td>
    <td>Fleischpflanzerl</td>
  </tr>
</table>

</body>
</html>
```

Um Bezüge zwischen Datenzellen und Kopfzellen nach dem obigen Beispiel herzustellen, müssen Sie in den Kopfzellen das Attribut `scope=` notieren (*scope = Geltungsbereich*). Dabei sind folgende Angaben erlaubt:

- `scope="col"` ist für Spaltenüberschriften gedacht und bedeutet, dass der Inhalt der Kopfzelle bei allen Zellen dieser Spalte wiederholt wird (*col = Spalte*).
- `scope="row"` ist für Zeilenüberschriften (z.B. links als erste Tabellenspalte) gedacht und bedeutet, dass der Inhalt der Kopfzelle bei allen Zellen dieser Zeile wiederholt wird (*row = Zeile*).
- `scope="colgroup"` ist für Spaltenüberschriften von Spaltengruppen gedacht und bedeutet, dass der Inhalt der Kopfzelle bei allen Zellen aller Spalten der Spaltengruppe wiederholt wird.

(*colgroup* = *Spaltengruppe*). Voraussetzung ist, dass Spaltengruppen mit Hilfe von `<colgroup>` vordefiniert werden.

- `scope="rowgroup"` ist für Spaltenüberschriften von Zeilengruppen gedacht und bedeutet, dass der Inhalt der Kopfzelle bei allen Zellen aller Zeilen der Zeilengruppe wiederholt wird (*rowgroup* = *Zeilengruppe*). Voraussetzung ist, dass Zeilengruppen in Form von Kopf, Körper und Fuß definiert werden.

### 8.7.5 Tabellen-Kategorien definieren

Bei komplexeren Tabellen können Sie gemeinsame Kategorienamen für Zellen vergeben. Dazu eine Grafik zur Veranschaulichung (das Beispiel stammt ursprünglich vom W3-Konsortium).

|                | Speisen | Hotel  | Fahrtkosten | Summen |
|----------------|---------|--------|-------------|--------|
| <b>Hamburg</b> |         |        |             |        |
| 12.08.98       | 37.74   | 112.00 | 45.00       |        |
| 13.08.98       | 27.28   | 112.00 | 45.00       |        |
| Summen         | 65.02   | 224.00 | 90.00       | 379.02 |
| <b>Berlin</b>  |         |        |             |        |
| 14.08.98       | 96.25   | 109.00 | 36.00       |        |
| 15.08.98       | 35.00   | 109.00 | 36.00       |        |
| Summen         | 131.25  | 218.00 | 72.00       | 421.25 |
| <b>Gesamt</b>  | 196.27  | 442.00 | 162.00      | 800.27 |

8.19: Dieser Beispieltabelle sollen Kategorienamen für Zellen zugewiesen werden.

In einem interaktiven Sprachausgabe-/Sprachbefehlssystem könnte ein Anwender zu einer solchen Tabelle etwa Fragen stellen wie:

- Wie viel habe ich insgesamt für Speisen ausgegeben?
- Wie viel habe ich am 13.8.1998 für Speisen ausgegeben?
- Wie viel habe ich in Hamburg insgesamt ausgegeben?

Um in etwa zu entscheiden, wie viel am 13.08.1998 für Speisen ausgegeben wurden, muss der Computer wissen, welche Zellen zum Typ »Datum« gehören und welche zum Typ »Speisen«. Solche Kategorienamen können Sie für Tabellenzellen in HTML vergeben.

Einige Beispiele (nur Details):

```
<th id="a4" axis="Datum">12.08.1998</th>
<th id="a5" axis="Datum">13.08.1998</th>
<th id="b1" axis="Ausgaben">Speisen</th>
<th id="b2" axis="Ausgaben">Hotel</th>
```

Das Beispiel ist keine echte Tabelle. Es zeigt nur die Definition einiger Kopfzellen (vergleiche Grafik weiter oben). Mit `id=` erhalten die Zellen unterschiedliche individuelle Namen, im Beispiel `a4` und `a5`. Das ist für die spätere Referenzierung wichtig. Ferner erhalten die Zellen aber auch noch einen gemeinsamen Kategorienamen. Solche Kategorienamen können Sie mit dem Attribut `axis=` vergeben (*axis* = *Achse*).

Durch die Kategorienamen hat ein elektronisches System alle Informationen, um Fragen von der Art »wie viel habe ich am Datum=13.8.1998 für Ausgaben=Speisen ausgegeben?« zu beantworten.

## 9 Frames – mehrere Fenster

### 9.1 Framesets und Frames definieren

#### 9.1.1 Allgemeines zu Frames

Mit Hilfe von Frames können Sie den Anzeigebereich des Browsers in verschiedene, frei definierbare Segmente aufteilen. Jedes Segment kann eigene Inhalte enthalten. Die einzelnen Anzeigesegmente (also die Frames) können wahlweise einen statischen Inhalt (= »non scrolling regions«) oder einen wechselnden Inhalt haben. Verweise in einem Frame können Dateien aufrufen, die dann in einem anderen Frame angezeigt werden.

Frames sind kein weiteres Element, um typische Aufgaben der Textverarbeitung zu bewältigen, sondern ein Element, das die spezifischen Eigenschaften der Bildschirmanzeige konsequent nutzt. Frames eröffnen völlig neue Möglichkeiten, um Information hypertextuell (d.h. nicht-linear) aufzubereiten.

Frames werden ab Netscape 2.0 und ab MS Internet Explorer 3.0 unterstützt. Seit HTML 4.0 gehören die Frames auch zum HTML-Standard. Dort allerdings nicht zur HTML-Variante »Strict«, sondern stattdessen mit einer eigenen Variante »Frameset« ausgestattet. Die folgende Abbildung zeigt, wie Frames im Anzeigefenster des Browsers wirken:



9.1: Der typische Aufbau einer Web-Seite mit drei Frames.

Das Bild stellt den Inhalt des gesamten Anzeigefensters des Browsers schematisch dar. Das Anzeigefenster ist in drei unabhängige Bereiche aufgeteilt. Die Verweise links und unten können beispielsweise immer eingeblendet bleiben, während sich der Inhalt des Hauptfensters je nach ausgewähltem Verweis ändern kann. Dabei werden im Beispiel der Abbildung immer drei verschiedene HTML-Dateien gleichzeitig angezeigt: links und unten immer die gleiche Datei, im Hauptfenster jeweils eine wechselnde Datei, je nach ausgewähltem Verweis.

In jedem Frame stehen alle Anzeige-Features zur Verfügung – so ist es z.B. denkbar, in einem Frame Textinformation anzuzeigen, während in einem anderen Frame gleichzeitig ein passendes Video abläuft.

Durch den Einsatz von Frames wachsen die Gestaltungsmöglichkeiten außerordentlich. Frames stellen an das Design von HTML-Seiten aber auch besonders hohe Ansprüche und haben einige nicht zu bestreitende Nachteile. Einige Tipps, die Sie beachten sollten, finden Sie im Abschnitt 9.5 *Frames als Mittel für Seitenlayouts*.

### 9.1.2 Grundgerüst einer HTML-Datei mit Framesets

Um Frames zu definieren, brauchen Sie eine spezielle HTML-Datei, in der ein Frameset definiert wird. Das Frameset bestimmt die Fensteraufteilung. Für eine solche Datei ist ein anderes Grundgerüst erforderlich als das sonst übliche Grundgerüst von HTML-Dateien.

Schema:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<frameset ...> <!-- Frameset-Definition -->
  <frame ...> <!-- Frame-Fenster-Definition -->
  <noframes>
    Wird angezeigt, wenn der Browser
    keine Frames anzeigen kann
  </noframes>
</frameset>
</html>
```

Die Unterschiede zu anderen HTML-Dateien beginnen bei der Dokumenttyp-Angabe. Bei Dateien, in denen Sie Framesets und Frames definieren, müssen Sie die oben gezeigte Dokumenttyp-Angabe zur HTML-Variante »Frameset« notieren.

Ein weiterer wichtiger Unterschied zu anderen HTML-Dateien ist, dass Dateien mit Frameset-Definitionen kein `body`-Element besitzen. Anstelle des `body`-Elements, also nach dem abschließenden `</head>`-Tag für den Dateikopf, werden die Frames definiert. Dabei können Sie:

- Framesets definieren
- Frames zu einem Frameset definieren
- Noframes-Bereich definieren

Mit Framesets bestimmen Sie die Aufteilung der Frame-Fenster, mit Frames die Datenquellen der einzelnen Frame-Fenster, und der Noframes-Bereich ist für Browser gedacht, die keine Frames anzeigen können, oder bei denen die Anzeige von Frames deaktivierbar ist und vom Anwender deaktiviert wurde.

**Beachten Sie:** Der Titel (`<title>...</title>`), den Sie in der Datei mit der Frameset-Definition angeben, wird während der gesamten Dauer des Framesets angezeigt, auch wenn andere Dateien innerhalb des Framesets aufgerufen werden. Leider gibt es bislang keine Möglichkeit,

den Titel zu aktualisieren. Wählen Sie in der Datei, die die Frameset-Definitionen enthält, deshalb einen allgemeinen, aussagekräftigen Titel, der für das gesamte Projekt Gültigkeit hat.

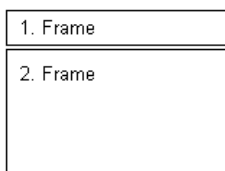
Der URI der Datei mit der Frameset-Definition, also beispielsweise *http://www.ihr-guter-name.de/*, bleibt in der gleichen Form in der Adresszeile des Browsers stehen, auch wenn der Anwender durch Navigieren innerhalb des Framesets andere Seiten des Projekts in eines der Frame-Fenster lädt. Mit anderen Worten: Ein Anwender kann immer nur die Startseite als Lesezeichen setzen, nicht eine bestimmte Seite, die in einem Frame angezeigt wird.

### 9.1.3 Framesets definieren

Beim Definieren von Framesets bestimmen Sie, wie das Anzeigefenster des Browsers in verschiedene Frame-Fenster aufgeteilt werden soll. Dabei müssen Sie sich das Anzeigefenster wie den leeren Rahmen einer Tabelle vorstellen. Damit die Tabelle Gestalt annimmt, definieren Sie Reihen und Spalten für die Frame-Fenster. Ebenso, wie es möglich ist, Tabellen in HTML zu verschachteln, ist das auch bei Framesets möglich. Anstelle eines Frames können Sie auch ein untergeordnetes Frameset notieren. Auf diese Weise können Sie beliebige Fensteraufteilungen erreichen.

#### **Detailbeispiel 1**

```
<frameset rows="20%,80%">
  <!-- Dadurch ergeben sich zwei Frame-Fenster,
        deren Inhalt hier bestimmt wird -->
</frameset>
```

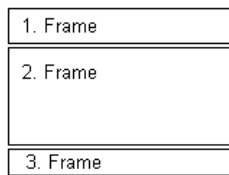


#### 9.2: Detailbeispiel 1: Zwei Frames untereinander mit waagrechter Teilung.

Im einleitenden Tag `<frameset...>` bestimmen Sie die Aufteilung. Durch `rows=` teilen Sie das Anzeigefenster in Reihen auf (*rows = Reihen*). Dahinter bestimmen Sie, wie die Aufteilung genau aussehen soll. Im Beispiel wird mit Hilfe von `rows="20%,80%"` eine Aufteilung in zwei Reihen erzwungen, wobei die obere Reihe 20% des Anzeigefensters in Anspruch nimmt, die untere 80%. Trennen Sie die Angaben zur Aufteilung durch Kommata.

#### **Detailbeispiel 2**

```
<frameset rows="100,*,60%">
  <!-- Dadurch ergeben sich drei Frame-Fenster,
        deren Inhalt hier bestimmt wird -->
</frameset>
```



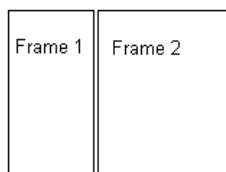
9.3: Detailbeispiel 2: Drei Frames untereinander mit waagrechtter Teilung.

Mit der Angabe `rows="100,*,60"` werden drei untereinander liegende Reihen für Frame-Fenster erzeugt. Die obere Reihe wird genau 100 Pixel hoch, die untere Reihe genau 60 Pixel hoch, und die mittlere Reihe erhält den Rest des Anzeigefensters, abhängig von der Größe des Anzeigefensters beim Anwender.

Mit Zahlenangaben, die ein Prozentzeichen enthalten, wird die Aufteilung also prozentual (relativ zur Größe des Anzeigefensters) interpretiert. Bei Zahlenangaben ohne Prozentzeichen wird die Angabe als absoluter Pixelwert interpretiert. Mit dem Stern-Zeichen `*` definieren Sie Bereiche von relativer Größe, abhängig von den übrigen Angaben. Sie können auch vier oder mehrere Reihen definieren. So definieren Sie beispielsweise mit `rows="10%,25%,30%,25%,10%"` fünf Reihen.

### Detailbeispiel 3

```
<frameset cols="200,*">
  <!-- Dadurch ergeben sich zwei Frame-Fenster,
        deren Inhalt hier bestimmt wird -->
</frameset>
```



9.4: Detailbeispiel 3: Zwei Frames nebeneinander mit senkrechter Teilung.

Mit dem Attribut `cols=` teilen Sie das Anzeigefenster in Spalten auf (`cols = columns = Spalten`). Dahinter bestimmen Sie, wie die Aufteilung genau aussehen soll. Im Beispiel wird mit Hilfe von `cols="200,*"` eine Aufteilung in zwei Spalten erzwungen, wobei die linke Spalte 200 Pixel des Anzeigefensters in Anspruch nimmt, die rechte den Rest. Bei `cols=` sind die gleichen Angaben möglich wie bei `rows=`.

### Detailbeispiel 4:

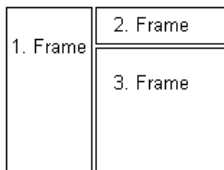
```
<frameset cols="40%,60%">
  <!-- Dadurch ergeben sich zwei
        Frame-Fenster, deren Inhalt hier bestimmt wird
        Das erste Frame-Fenster wird normal definiert,
        anstelle des zweiten folgt jedoch das folgende
```



```

untergeordnete Frameset: -->
<frameset rows="20%,80%">
  <!-- Noch mal zwei Frame-Fenster, deren
    Inhalt hier bestimmt wird -->
</frameset>
</frameset>

```



9.5: Detailbeispiel 4: Drei Frames mit waagrechter und senkrechter Teilung.

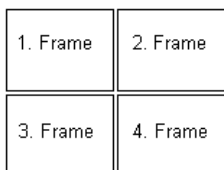
In diesem Beispiel werden Reihen und Spalten für eine komplexere Aufteilung von Frame-Fenstern definiert. Bei solchen Aufteilungen müssen Sie zunächst das gesamte Fenster im Auge behalten. Dieses Fenster wird im Beispiel zunächst mit `cols="40%,60%"` in zwei Spalten aufgeteilt. Das erste Frame-Fenster für die linke Spalte (im obigen Beispiel nicht explizit definiert, dort steht nur der längere Kommentar) wird ein ganz normales Frame-Fenster mit Inhalt. Anstelle des zweiten Frame-Fensters für die rechte Spalte wird jedoch ein neues, untergeordnetes Frameset definiert. Mit der Angabe `rows="20%,80%"` erhält es eine Aufteilung in zwei Reihen, also für zwei untereinander liegende Frame-Fenster, wobei das obere 20% der zur Verfügung stehenden Fläche einnimmt und das untere 80%.

### Detailbeispiel 5

```

<frameset rows="50%,50%">
  <frameset cols="50%,50%">
    <!-- zwei Frames, deren Inhalt hier bestimmt wird -->
  </frameset>
  <frameset cols="50%,50%">
    <!-- noch mal zwei Frames, deren Inhalt
      hier bestimmt wird -->
  </frameset>
</frameset>

```



9.6: Detailbeispiel 5: Vier Frames in Schachbrettanordnung.

Das Beispiel zeigt eine komplexere Aufteilung in vier Bereiche. Da in einem solchen Fall alle Bereiche gleich groß sind, ist es egal, ob Sie mit der Definition der Spalten oder Reihen beginnen.

Wichtig ist die durchgängige Logik. Wenn Sie – wie im Beispiel – mit der Definition der Reihen beginnen, müssen Sie anstelle von Frames untergeordnete Framesets definieren, in denen Sie jede der beiden Reihen in zwei Spalten aufteilen. Erst innerhalb dieser untergeordneten Framesets werden dann die eigentlichen Frames definiert.

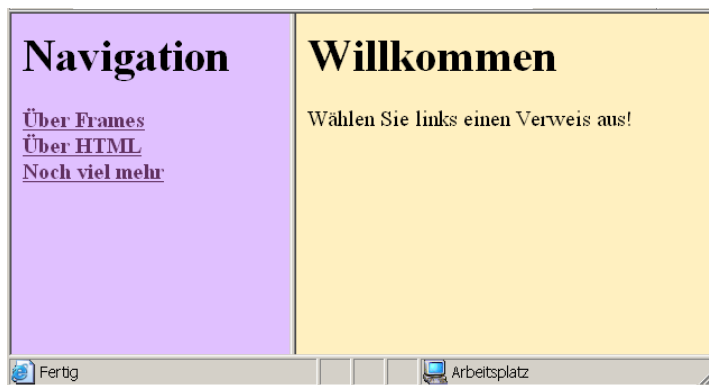
**Beachten Sie:** Framesets sollten immer so definiert sein, dass das gesamte Anzeigefenster abgedeckt wird. Verwenden Sie dazu Prozentangaben, die in der Summe 100 ergeben, oder das Stern-Zeichen \*.

Neben der prozentualen Aufteilung ist noch eine weitere relative Angabe zur Frame-Fenster-Aufteilung erlaubt. Sie können das relative Verhältnis bei `rows=` oder `cols=` auch durch Zahlen, gefolgt von einem Stern-Zeichen, bestimmen.

Mit einer Aufteilung wie `rows="1*,5*,3*"` definieren Sie drei Reihen, also Platz für drei untereinander liegende Frame-Fenster, wobei das oberste Frame-Fenster ein Neuntel einnimmt, das zweite fünf Neuntel und das dritte drei Neuntel (also ein Drittel). Die »9« ergibt sich durch Zusammenzählen von 1, 3 und 5, wodurch das relative Verhältnis ermittelt wird.

#### 9.1.4 Frames zu einem Frameset definieren

Nachdem Sie durch die Definition von Framesets die gewünschte Aufteilung des Anzeigefensters bestimmt haben, geben Sie mit der Definition der Frames an, welche Inhalte in den einzelnen Frame-Fenstern zunächst angezeigt werden sollen.



9.7: So sieht das nachfolgende Beispiel mit drei Dateien und zwei Frames im Browser aus.

Beispiel Teil 1 – Datei mit Frameset:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
    "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
    <frame src="verweise.htm" name="Navigation">
```

```

<frame src="startseite.htm" name="Daten">
<noframes>
  Ihr Browser kann diese Seite leider nicht anzeigen!
</noframes>
</frameset>
</html>

```

Beispiel Teil 2 – Datei *verweise.htm* für linkes Frame-Fenster:

```

<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#E0C0FF" text="#000000" link="#804080" vlink="#603060"
alink="#804080">
<h1>Navigation</h1>
<p>
<a href="../index.htm" target="Daten">
  <b>Über Frames</b>
</a><br>
<a href="../../index.htm" target="Daten">
  <b>Über HTML</b>
</a><br>
<a href="../../../index.htm" target="Daten">
  <b>Noch viel mehr</b>
</a>
</p>
</body>
</html>

```

Beispiel Teil 3 – Datei *startseite.htm* für rechtes Frame-Fenster:

```

<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#FFF0C0" text="#000000" link="#A00000" vlink="#600000"
alink="#A00000">
<h1>Willkommen</h1>
<p>Wählen Sie links einen Verweis aus!</p>
</body>
</html>

```

Mit je einem `<frame>`-Tag definieren Sie innerhalb eines Framesets die zugehörigen Frames (*frame* = *Rahmen*). Theoretisch können Sie das `<frame>`-Tag ohne weitere Attribute notieren. In diesem Fall zeigt der Browser in dem entsprechenden Frame-Fenster eine leere Fläche an. Normalerweise werden Sie jedoch die Frame-Fenster mit Inhalt füllen wollen. Dazu dient das Attribut `src=` (*src* = *source* = *Quelle*). Bei der Wertzuweisung an dieses Attribut können Sie die Datei oder Datenquelle angeben, die im entsprechenden Frame-Fenster zunächst angezeigt werden soll. Dabei gelten die Regeln zum Referenzieren in HTML – mit einer Ausnahme: Sie dürfen keine Anker innerhalb der Datei mit Framesets adressieren. Eine Wertzuweisung wie `src="#weiter_unten"` ist also nicht erlaubt – widerspricht allerdings sowieso jeder Logik.

Sie können also andere HTML-Dateien des eigenen Projekts in das Frame-Fenster laden, aber auch andere Dateitypen wie z.B. Grafiken. Auch Dateien oder Datenquellen von entfernten Servern können Sie – mit absoluter `http://`-Adressierung beispielsweise – in ein Frame-Fenster laden. Doch dabei ist höchste Vorsicht geboten: Damit machen Sie sich fremde Inhalte zu Eigen und können sich schnell juristischen Ärger einhandeln.

Ferner sollten Sie gleich für jedes definierte Frame-Fenster mit `name=` einen Namen vergeben. Diese Namen brauchen Sie, um Verweise zu definieren, die in diesem Frame-Fenster angezeigt werden sollen. Ferner sind die Namen wichtig, wenn Sie zusätzlich JavaScript einsetzen und damit auf einzelne Frame-Fenster zugreifen wollen.

Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (`_`), den Bindestrich (`-`), den Doppelpunkt (`:`) oder den Punkt (`.`). Im Hinblick auf JavaScript darf der Name sogar nur Buchstaben, Ziffern und den Unterstrich (`_`) enthalten. Groß- und Kleinschreibung werden bei Sprachen wie JavaScript ebenfalls unterschieden.

Definieren Sie für jede »Zelle« Ihrer durch das Frameset definierten »Tabelle« ein Frame-Fenster. Im obigen Beispiel wird ein Frameset für zwei Frame-Fenster definiert, mit der Aufteilung, die durch die Angabe `cols="250,*">` bestimmt wird. Ins linke Frame-Fenster, also dem mit 250 Pixeln Breite, soll im Beispiel eine Datei namens `verweise.htm` geladen werden, und ins rechte Frame-Fenster eine Datei namens `startseite.htm`. So wie im Beispiel bei `src=` notiert, müssen sich beide Dateien im gleichen Verzeichnis befinden.

Die beiden HTML-Dateien selbst sind gewöhnliche HTML-Dateien. An der Datei `verweise.htm` können Sie bei den darin notierten Verweisen sehen, wozu die Namen der Frame-Fenster gut sind: Mit `target="Daten"` wird dort bei den Verweisen bestimmt, dass das Verweisziel in dem Frame-Fenster angezeigt wird, das bei der Frame-Fenster-Definition mit `name="Daten"` den entsprechenden Namen erhalten hat. Weitere Einzelheiten zu diesem Thema finden Sie im Abschnitt 9.3 *Verweise bei Frames*.

**Beachten Sie:** Das `frame`-Element besteht nur aus dem Standalone-Tag `<frame>`. Wenn Sie XHTML-konform arbeiten, müssen Sie das `frame`-Element als inhaltsleer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<frame />`. Weitere Informationen dazu im Kapitel 14.1.11 zu *XHTML und HTML*.

### 9.1.5 Noframes-Bereich definieren

Für Anwender, deren Browser keine Frames unterstützen, können Sie Ihr Projekt so gestalten, dass solche Anwender dennoch Daten aufrufen können. Noframes-Bereiche haben dabei zwei Aufgaben: Zum einen können sie in der Datei stehen, in der das Frameset definiert ist. Dort können Sie für Browser, die keine Frames anzeigen können oder bei denen der Anwender die Anzeige von Frames abgeschaltet hat, einen alternativen Inhalt anbieten. Es kann sich um beliebige, mit HTML strukturierte Inhalte handeln.

Darüber hinaus können Sie Noframes-Bereiche aber auch in HTML-Dateien verwenden, die innerhalb des Framesets als Inhalte in Frame-Fenstern angezeigt werden sollen. Browser, bei denen die Frames angezeigt werden, sollten den Inhalt solcher Noframes-Bereiche unterdrücken. Browser, bei denen die Frames nicht angezeigt werden können oder keine Frames angezeigt werden, zeigen den Inhalt an. Auf diese Weise können Sie z.B. eine einfache Alternativ-Navigation für nicht framefähige Browser anbieten. Das folgende Beispiel zeigt die Verwendungsweisen für Noframes-Bereiche.

Beispiel Teil 1 – Datei mit Frameset:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
    "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
  <frame src="verweise2.htm" name="Navigation">
  <frame src="startseite2.htm" name="Daten">
  <noframes>
    <h1>Willkommen!</h1>
    <p>Dieses Projekt verwendet Frames. Bei Ihnen
      werden keine Frames angezeigt.</p>
    <p>W&auml;hlen Sie einen der Verweise aus:<br>
    <a href="startseite2.htm"><b>Startseite</b></a><br>
    <!-- weitere Verweise usw. --></p>
  </noframes>
</frameset>
</html>
```

Beispiel Teil 2 – Datei *startseite2.htm* für rechtes Frame-Fenster:

```
<!DOCTYPE HTML PUBLIC
    "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#FFF0C0" text="#000000" link="#A00000" vlink="#600000"
  alink="#A00000">
```

```

<h1>Willkommen</h1>
<p>Willkommen Sie links einen Verweis aus!</p>

<noframes>
<p>Dieses Projekt verwendet Frames. Bei Ihnen werden keine Frames angezeigt:</p>
<p><a href="frames2.htm">zur Startseite</a></p>
</noframes>

</body>
</html>

```

Das Beispiel zeigt die Datei mit der Frameset-Definition und die Datei, die im rechten Frame-Fenster angezeigt wird. Die Datei `verweise.htm`, die ins linke Frame-Fenster geladen wird, ist in diesem Zusammenhang hier unerheblich und wird deshalb nicht nochmal dargestellt.

Innerhalb von Dateien, die Framesets definieren, sollte ein Noframes-Bereich, markiert durch `<noframes>` und `</noframes>`, am Ende des äußersten Framesets vor dem letzten abschließenden `</frameset>`-Tag notiert werden. Im obigen Beispiel wird nur ein einfaches Frameset definiert, das aus zwei Frame-Fenstern besteht. In diesem Fall werden also die beiden Frame-Fenster notiert und unterhalb davon, vor dem abschließenden `</frameset>`, der Noframes-Bereich.

Innerhalb des Noframes-Bereichs können Sie alles notieren, was Sie sonst auch zwischen `<body>` und `</body>` notieren können, also z.B. Überschriften, Textabsätze, Tabellen, Grafikreferenzen usw. Da es sich bei Browsern, die keine Frames anzeigen können, in der Regel um sehr einfach gestrickte Browser handelt oder um solche, die unter nichtgrafischen Oberflächen laufen, brauchen Sie keinen übermäßigen Aufwand für die Optik des Inhalts im Noframes-Bereich zu betreiben. Eine saubere Strukturierung genügt.

Im obigen Beispiel ist innerhalb des Noframes-Bereichs der Datei mit der Frameset-Definition eine Willkommens-Überschrift notiert, ein Hinweis, dass es sich um ein Projekt mit Frames handelt, und es werden Verweise angeboten. Beispielsweise der Verweis auf die Seite `startseite2.htm`, die bei Anzeige von Frames im rechten Frame-Fenster erscheint. Ferner könnten Sie an dieser Stelle all jene Verweise anbieten, die bei Anzeige von Frames in der links angezeigten Datei `verweise2.htm` stehen. Auf diese Weise können Anwender ohne framefähigen Browser dennoch alle Verweise ausführen, die sonst im Frameset angeboten werden.

In der Datei `startseite2.htm` taucht im obigen Beispiel jedoch auch ein Noframes-Bereich auf. Für Browser, die keine Frames anzeigen können, können Sie in solchen Bereichen Inhalte anbieten, die sonst über das Frameset erledigt werden – beispielsweise einen Rückverweis auf die Startseite. Diese Möglichkeit wird auch im Beispiel benutzt: Durch den Rückverweis auf die Datei mit den Frameset-Definitionen, die ja auch einen Noframes-Bereich enthält und dort Verweise auf die übrigen Projektseiten anbietet, können Sie eine vollständige, wenn auch etwas simple Alternativ-Navigation für Anwender ohne framefähigen Browser realisieren.

**Beachten Sie:** Noframes-Bereiche in Projektdateien außerhalb der Datei mit Frameset-Definitionen sind allerdings mit Vorsicht zu genießen, denn leider werden sie nicht nur von nicht framefähigen Browsern angezeigt, sondern auch von älteren framefähigen. So zeigt beispielsweise Netscape 4.x den Noframes-Bereich in der Datei `startseite2.htm` des obigen Beispiels gnadenlos an und verwirrt damit Anwender, bei denen das Frameset angezeigt wird. Internet

Explorer 5.x und Netscape 6.x unterdrücken dagegen Noframes-Bereiche in solchen Projektdateien.

Wenn Sie in Projektdateien, die in einem Frame-Fenster angezeigt werden, Noframes-Bereiche notieren, müssen Sie für solche Projektdateien die HTML-Variante »Transitional« wählen, denn in der »Strict«-Variante gibt es keine Frames und auch keine Noframes-Bereiche.

### 9.1.6 Langbeschreibung für Frame-Fenster

Sie können bei der Definition von Frames angeben, wo eine ausführliche Beschreibung des Inhalts des betreffenden Frame-Fensters zu finden ist. Browser, die zwar Frames erkennen, aber bei denen die Anzeige von Frames nicht möglich ist, z.B. akustische Ausgabesysteme, können z.B. das bei der Langbeschreibung angegebene Ziel ausgeben. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
  <frame src="verweise.htm" name="Navigation">
  <frame src="startseite.htm" name="Daten"
    longdesc="startseite_extra.htm">
  <noframes>
    Ihr Browser kann diese Seite leider nicht anzeigen!
  </noframes>
</frameset>
</html>
```

Erläuterung: Mit dem Attribut `longdesc=` in einem `<frame>`-Tag können Sie auf eine Stelle verweisen, wo der Inhalt dieses Frame-Fensters in einer längeren Beschreibung zusammengefasst wird. Erlaubt ist ein URI, d.h. bei der Angabe gelten die Regeln zum Referenzieren in HTML.

## 9.2 Eigenschaften von Frames

### 9.2.1 Bildlaufleisten erzwingen/verhindern

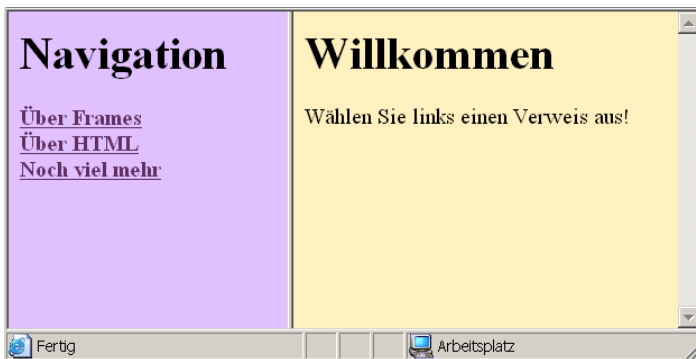
Der Web-Browser verwaltet die Scrollbars (Bildlaufleisten) eines jeden Frame-Fensters normalerweise automatisch. Wenn kein Blättern erforderlich ist, um den gesamten Fensterinhalt anzuzeigen, erhält das Fenster keine Scrollbars. Wenn der Inhalt größer ist, werden die Frame-Fenster automatisch mit Scrollbars ausgestattet. Sie können jedoch erzwingen, dass ein Frame-Fenster in jedem Fall Scrollbars bzw. keine Scrollbars hat. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
```

```

<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
  <frame src="verweise.htm" name="Navigation" scrolling="no">
  <frame src="startseite.htm" name="Daten" scrolling="yes">
</frameset>
  <p>Dieses Projekt verwendet Frames. Bei
    Ihnen werden keine Frames angezeigt.</p>
</frameset>
</html>

```



9.8: Rechts ist eine (im Bild deaktivierte) Bildlaufleiste für den Frame eingeblendet, egal, ob sie benötigt wird oder nicht.

Durch die Angabe `scrolling="yes"` in einem `<frame>`-Tag erzwingen Sie, dass das Anzeigefenster dieses Frame-Fensters in jedem Fall Scrollbars besitzt. Durch `scrolling="no"` verhindern Sie, dass das Frame-Fenster Scrollbars erhält, selbst wenn der Inhalt länger ist. Ebenfalls möglich ist die Angabe `scrolling="auto"`, die jedoch mit der Voreinstellung (Scrollbars bei Bedarf anzeigen) identisch ist und daher weggelassen werden kann.

**Beachten Sie:** Wenn Sie das Scrollen des Fensterinhalts verhindern, können Inhalte, die größer sind als das Fenster, nicht vollständig angezeigt werden. Verwenden Sie `scrolling="no"` daher nur, wenn Sie sicher sind, dass der Fensterinhalt vollständig in das Frame-Fenster passt. Die Angabe ist zum Beispiel sinnvoll, wenn in einem Frame-Fenster dauerhaft ein Logo angezeigt werden soll und sonst nichts. Bei größeren Inhalten, vor allem in Frame-Fenstern mit relativer Größe, sollten Sie mit dem Ausschalten der Scrollbars vorsichtig sein – bedenken Sie, dass nicht alle Anwender die gleiche Bildschirmauflösung haben wie Sie.

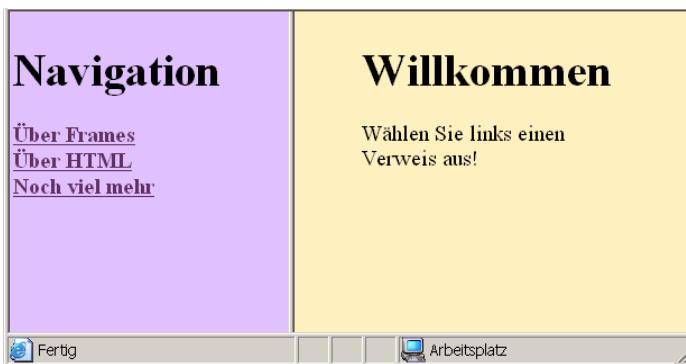
### 9.2.2 Mindestabstände zwischen Fensterrand und Fensterinhalt

Sie haben die Möglichkeit, den Abstand zwischen Fensterrand und dem angezeigten Inhalt eines Frame-Fensters genau zu justieren. Auf diese Weise können Sie zum Beispiel breitere Seitenränder in einem Frame-Fenster erzeugen. Wenn Sie einen festen Inhalt, etwa ein Logo, in einem



Frame-Fenster anzeigen wollen, können Sie das Logo durch Justage der Abstände exakt in einem Frame-Fenster platzieren. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
  <frame src="verweise.htm" name="Navigation"
    scrolling="no" marginwidth="3" marginheight="30">
  <frame src="startseite.htm" name="Daten"
    marginwidth="60" marginheight="30">
</frameset>
  <p>Dieses Projekt verwendet Frames. Bei Ihnen
    werden keine Frames angezeigt.</p>
</frameset>
</html>
```



9.9: In beiden Frames sind unterschiedliche Abstände von Inhalt zum Fensterrand gesetzt.

Durch das Attribut `marginwidth= [Pixel]` in einem `<frame>`-Tag bestimmen Sie den Abstand zwischen rechtem bzw. linkem Fensterrand und dem Fensterinhalt in diesem Frame-Fenster (*margin* = Rand, *width* = Breite). Durch das Attribut `marginheight= [Pixel]` bestimmen Sie den Abstand zwischen oberem bzw. unterem Fensterrand und dem Fensterinhalt (*height* = Höhe). Linke und rechte Ränder sind immer gleich groß, ebenso obere und untere.

**Beachten Sie:** Sie können zu beiden Werten Angaben machen, aber auch nur zu einem der beiden. Mit den Angaben `marginwidth="0" marginheight="0"` sollte der Fensterinhalt exakt in der linken oberen Ecke beginnen. Leider fügt Netscape 4.x bei der Anzeige doch immer noch ein Pixel »Seitenrand« ein. Als Alternative dazu bleibt nur, den Inhalt mit Hilfe von CSS Style-sheets absolut zu positionieren mit `top:0px` und `left:0px`.

### 9.2.3 Unveränderbare Fenstergröße

Normalerweise kann der Anwender in einem angezeigten Frameset die Größe der einzelnen Frame-Fenster verändern, indem er die Maus auf die Rahmen zwischen Frame-Fenstern positioniert und durch »Ziehen« die Größenverhältnisse der Frame-Fenster einstellt. In vielen Fällen ist das auch sinnvoll, und Sie sollten dem Anwender diese Freiheit lassen. Wenn Sie aber beispielsweise ein Frame-Fenster mit einem fixen Inhalt, haben, etwa einem Logo, können Sie verhindern, dass der Anwender die Fenstergröße verändern kann. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
    "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
    <frame src="verweise.htm" name="Navigation"
        scrolling="no" noresize>
    <frame src="startseite.htm" name="Daten">
</frameset>
    <p>Dieses Projekt verwendet Frames. Bei Ihnen
        werden keine Frames angezeigt.</p>
</frameset>
</html>
```

Durch das Attribut `noresize` in einem `<frame>`-Tag verhindern Sie, dass der Anwender die Größe des Frame-Fensters verändern kann (*noresize* = *no resize* = *keine Größenveränderung*).

**Beachten Sie:** Nicht nur das Frame-Fenster, für das Sie das Attribut notieren, kann dann nicht mehr in der Größe verändert werden, sondern auch alle benachbarten Frame-Fenster. Im obigen Beispiel kann der Anwender also weder das linke noch das rechte Frame-Fenster verändern. Das ist eigentlich logisch, aber Sie sollten diese Folgewirkung berücksichtigen, wenn Sie erwägen, in einem Frame-Fenster das Verändern der Größe zu verhindern.

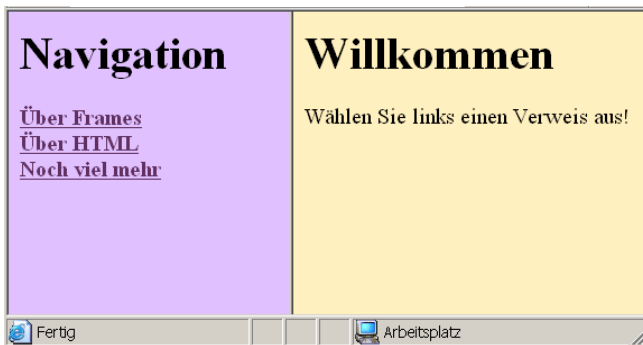
Wenn Sie **XHTML-konform** arbeiten wollen, müssen Sie das hier beschriebene Attribut in der Form `noresize="noresize"` notieren. Weitere Informationen dazu im Kapitel 14.1.11 über *XHTML und HTML*.

### 9.2.4 Rahmendicke/Unsichtbare Fensterrahmen

Sie können die normalerweise sichtbaren Rahmen zwischen Frame-Fenstern unterdrücken. Bei unsichtbaren Fensterrahmen wirken die Frame-Fenster wie nahtlos aneinander gefügte Flächen am Bildschirm. Leider gibt es hierbei bis auf den heutigen Tag Differenzen zwischen den Browsern, die dieses Feature seit langem unterstützen, und dem W3-Konsortium, das eine andere Variante durchsetzen will, an die sich die Browser aber nicht halten. Sie haben also die Auswahl, HTML-konform zu schreiben, und die weit verbreiteten Browser ignorieren Ihre Angaben, oder Sie schreiben für die Browser und verzichten auf die HTML-Konformität.

## Beispiel 1 (HTML-konform):

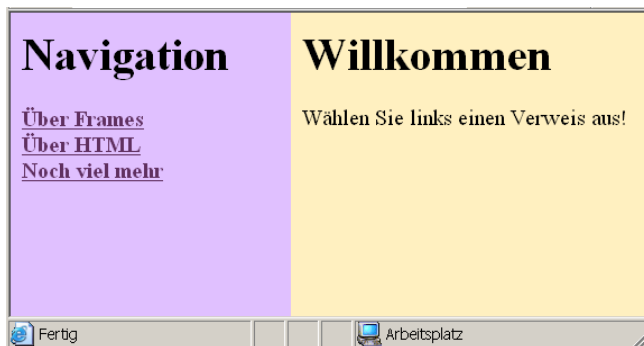
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
    "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
  <frame src="verweise.htm" name="Navigation"
    scrolling="no" frameborder="0">
  <frame src="startseite.htm" name="Daten">
  <noframes>
    <p>Dieses Projekt verwendet Frames. Bei Ihnen
      werden keine Frames angezeigt.</p>
  </noframes>
</frameset>
</html>
```



9.10: Mit den offiziellen HTML-Attributen bleibt eine Trennlinie zwischen den Frames sichtbar.

## Beispiel 2 (Browser-gerecht):

```
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*" frameborder="0"
  framespacing="0" border="0">
  <frame src="verweise.htm" name="Navigation" scrolling="no">
  <frame src="startseite.htm" name="Daten">
  <noframes>
    <p>Dieses Projekt verwendet Frames. Bei Ihnen
      werden keine Frames angezeigt.</p>
  </noframes>
</frameset>
</html>
```



9.11: Mit browserspezifischen Attributen verschwindet die Trennlinie zwischen den Frames.

Wenn Sie HTML-konform arbeiten wollen, notieren Sie in einem gewünschten `<frame>`-Tag das Attribut `frameborder=`. Mit `frameborder="0"` unterdrücken Sie die sichtbaren Rahmen zwischen dem betroffenen Frame-Fenster und allen seinen Nachbarfenstern, und mit `frameborder="1"` zeigen Sie die sichtbaren Rahmen explizit an (Letzteres entspricht allerdings der Voreinstellung).

Bei der Syntax, die von den Browsern unterstützt wird, werden die dazu nötigen Attribute im einleitenden `<frameset>`-Tag desjenigen Framesets notiert, für das die Angaben dann gelten – sie betreffen dann also alle Rahmen innerhalb des Framesets.

- `border= [Pixel]` ist Netscape-Syntax und bestimmt die Breite der Rahmen in Pixel. Erlaubt sind Zahlenwerte von 0 (keine Rahmen anzeigen) und größer (*border = Rahmen*).
- `frameborder= [1/0 oder yes/no]` bedeutet nach Microsoft-Syntax die Angabe, ob 3-D-Rahmen angezeigt werden sollen oder nicht. Dieses Attribut ist also gleichnamig mit dem HTML-standardkonformen Attribut, bedeutet jedoch etwas anderes.
- `framespacing= [Pixel]` ist Microsoft-Syntax und bestimmt den Abstand zwischen Frame-Fenstern, mithin also die Breite der Rahmen in Pixel (*spacing = Abstand*).

**Beachten Sie:** Nur mit `frameborder="0"` `framespacing="0"` (also mit beiden Angaben) unterdrücken Sie beim MS Internet Explorer die Rahmen konsequent. Wenn Sie nur `frameborder="0"` angeben, zeigt der Internet Explorer sichtbare Abstände zwischen den Frame-Fenstern an, allerdings nicht die üblichen 3-D-Rahmen, sondern flache Zwischenräume. Um die Rahmen beim Internet Explorer und bei Netscape zu unterdrücken, müssen Sie alle drei Angaben `frameborder="0"` `framespacing="0"` `border="0"` notieren (wie im obigen Beispiel 2).

Wenn Sie die Rahmen zwischen den Frame-Fenstern unterdrücken, hat der Anwender keine Möglichkeit, die Größe der Frame-Fenster zu verändern. Testen Sie Projekte mit rahmenlosen Frames deshalb möglichst unter mehreren Bildschirmauflösungen und mit unterschiedlich großem Anzeigefenster.

### 9.2.5 Farbige Fensterrahmen

Sie können erzwingen, dass Rahmen zwischen den Frame-Fenstern farbig dargestellt werden. Diese Angaben gehören jedoch nicht zum HTML-Standard und erzeugen ungültiges HTML. Ein Beispiel:

```
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*" bordercolor="#FFFF00">
  <frame src="verweise.htm" name="Navigation" scrolling="no">
  <frame src="startseite.htm" name="Daten">
  <noframes>
    <p>Dieses Projekt verwendet Frames. Bei Ihnen
      werden keine Frames angezeigt.</p>
  </noframes>
</frameset>
</html>
```

Mit den Attribut `bordercolor=` im obersten `<frameset>`-Tag können Sie eine Rahmenfarbe für alle Fensterrahmen bestimmen. Dabei gelten die Regeln zum Definieren von Farben in HTML (Kapitel 3.5).

## 9.3 Verweise bei Frames

### 9.3.1 Verweise zu anderen Frames

Wenn Sie mit Framesets arbeiten, werden Sie häufig in einem Frame-Fenster Verweise anbieten wollen, bei deren Anklicken das Verweisziel in einem anderen Frame-Fenster angezeigt werden soll.

Detailbeispiel Teil 1 – Frame-Definitionen:

```
<frameset cols="250,*">
  <frame src="verweise.htm" name="links">
  <frame src="startseite.htm" name="rechts">
  <noframes>
    Ihr Browser kann diese Seite leider nicht anzeigen!
  </noframes>
</frameset>
```

Detailbeispiel Teil 2 – Verweise in *verweise.htm*:

```
<a href="startseite.htm" target="rechts">Startseite</a><br>
<a href="news.htm" target="rechts">News</a><br>
<a href="feedback.htm" target="rechts">Feedback</a><br>
<a href="impressum.htm" target="rechts">Impressum</a><br>
```

Voraussetzung für Verweise zu anderen Frames ist, dass die Frame-Fenster eines Framesets Namen erhalten. Im obigen Beispiel wird ein Frameset mit zwei Frame-Fenstern definiert, die mit dem Attribut `name=` im `<frame>`-Tag die Namen `links` und `rechts` erhalten.

Für das Setzen eines Verweises zu einem anderen Frame-Fenster gelten beim `href`-Attribut wie üblich die Regeln zum Referenzieren in HTML. Damit das Verweisziel jedoch in einem anderen Frame-Fenster angezeigt wird, notieren Sie im einleitenden Verweis-Tag zusätzlich das Attribut `target=` (*target = Ziel*). Dahinter folgt der Name des Frame-Fensters, in dem das Verweisziel angezeigt werden soll.

**Beachten Sie:** Beim Namen, den Sie bei `name=` und dann bei `target=` angeben, soll zwar laut HTML-Standard nicht zwischen Groß- und Kleinschreibung unterschieden werden. Doch achten Sie in der Praxis unbedingt darauf, bei `target=` den Namen exakt so zu schreiben wie bei `name=` vergeben. Die Browser unterscheiden nämlich sehr wohl zwischen Groß-/Kleinschreibung – allein schon im Hinblick auf JavaScript!

Wenn Sie eine Datei mit Verweisen haben, deren Verweisziele alle in einem bestimmten anderen Frame-Fenster angezeigt werden sollen, sparen Sie Ressourcen, indem Sie eine Zielfensterbasis (Kapitel 6.1.4) definieren.

Für Verweise zu Zielen, die im gleichen Frame-Fenster angezeigt werden sollen wie der Verweis, benötigen Sie normalerweise keine zusätzliche Angabe im Verweis. Nur wenn Sie eine Zielfensterbasis in der Datei mit dem Verweis definiert haben, müssen Sie explizit angeben, dass die Datei im gleichen Frame-Fenster angezeigt werden soll. Dies erreichen Sie durch die Angabe `target="_self"` im einleitenden Verweis-Tag.

### 9.3.2 Verweise zum Beenden von Framesets

Wenn Sie mit Framesets arbeiten, werden Sie in einem Frame-Fenster manchmal auch Verweise anbieten wollen, bei deren Anklicken das Verweisziel nicht mehr in Ihrem Frameset angezeigt werden soll. Besonders bei Verweisen zu fremden Web-Projekten ist es dringend angebracht, das eigene Frameset zu beenden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#FFF0C0" text="#000000" link="#A00000" vlink="#600000"
alink="#A00000">
<h1>Willkommen</h1>
<p>Willkommen Sie links einen Verweis aus!
Oder probieren Sie mal diesen hier:<br>
<a href="http://selfaktuell.teamone.de/" target="_parent"><b>SELFHTML
aktuell</b></a></p>
</body>
</html>
```

Das Beispiel zeigt eine HTML-Datei, die innerhalb eines Framesets zum Einsatz kommt. Darin ist ein Verweis notiert, bei dessen Anklicken das Frameset »gesprengt« werden soll. Zu diesem Zweck ist im einleitenden Verweis-Tag `target="_parent"` notiert.

- `target="_parent"` bewirkt, dass das Verweisziel in dem Zustand des Anzeigefensters angezeigt wird, der vor dem Start Ihres Framesets aktuell war.
- `target="_top"` bewirkt, dass das Verweisziel in jedem Fall im gesamten Anzeigefenster angezeigt wird, selbst dann, wenn das aktuelle Frameset in ein Frame-Fenster eines übergeordneten Framesets geladen wurde.
- Weiterhin gibt es noch `target="_blank"`, wodurch das Verweisziel in einem neuen Instanzfenster des WWW-Browsers angezeigt wird. Das Anzeigefenster mit Ihrem Frameset bleibt im Hintergrund erhalten. Der Anwender kann wieder zu diesem Fenster wechseln, wenn er möchte.

**Beachten Sie:** Anstelle von `target="_blank"` können Sie auch `target="Halligalli"` oder sonst irgendeinen Unsinn notieren. Wenn es kein Fenster mit diesem Namen gibt, öffnen die Web-Browser das Verweisziel in einem neuen Browser-Fenster.

Das `target`-Attribut ist zwar nicht als *deprecated* gekennzeichnet, doch um es einzusetzen, müssen Sie in der betreffenden Datei die HTML-Variante »Transitional« verwenden. Der Grund ist, dass dieses Attribut für Verweise bei Verwendung von Frames gedacht ist und Frames eine eigene HTML-Variante haben, die von der Einstufung her der Variante »Transitional« entspricht (auf gut Deutsch: nicht der »reinen Lehre« entspricht).

## 9.4 Eingebettete Frames

### 9.4.1 Eingebettete Frames definieren

Eingebettete Frames werden vom MS Internet-Explorer ab Version 3.x interpretiert, von Netscape bis einschließlich Version 4.x dagegen nicht. Seit HTML 4.0 gehören sie zum HTML-Standard. Eingebettete Frames sind ein eigenständiges Gestaltungsmittel zur Informationsaufbereitung, das anders funktioniert als »normale« Frames. Eingebettete Frames erzeugen keine Aufteilung des Bildschirms, sondern sind ähnlich wie Grafiken Bereiche innerhalb einer HTML-Datei, in denen fremde Quellen, vor allem andere HTML-Dateien angezeigt werden können. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Eingebettete Frames definieren</title>
</head>
<body>

<h1>Fenstergucker</h1>
```

```

<p>Gucken Sie mal SELFHTML im Fenster an:</p>

<iframe src="../../index.htm" width="90%"
  height="400" name="SELFHTML_in_a_box">
<p>Ihr Browser kann leider keine eingebetteten
  Frames anzeigen: Sie können die eingebettete
  Seite über den folgenden Verweis aufrufen:
<a href="../../index.htm">SELFHTML</a></p>
</iframe>

</body>
</html>

```



9.12: SELFHTML erscheint innerhalb einer Web-Seite in einem eingebetteten Frame.

Für HTML-Dateien mit eingebetteten Frames gilt das gewöhnliche Grundgerüst einer HTML-Datei (Kapitel 3.2.1). Eingebettete Frames können dann im Textkörper der HTML-Datei an einer gewünschten Stelle vorkommen. Mit `<iframe...>` leiten Sie einen eingebetteten Frame ein (*iframe* = *inline frame* = *Rahmen im Textfluss*). Mit dem Attribut `src` bestimmen Sie, was in dem eingebetteten Frame angezeigt werden soll. Es kann sich um eine andere HTML-Datei oder eine beliebige andere lokale oder entfernte Datenquelle handeln. Bei der Wertzuweisung gelten die Regeln zum Referenzieren in HTML.

Ferner sollten Sie für den eingebetteten Frame mit `name` einen Namen vergeben. Diesen Namen brauchen Sie, um Verweise zu dem eingebetteten Frame zu definieren. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (`_`), den Bindestrich (`-`), den Doppelpunkt



(:) oder den Punkt (.). Im Hinblick auf JavaScript darf der Name sogar nur Buchstaben, Ziffern und den Unterstrich (\_) enthalten. Groß- und Kleinschreibung werden bei Sprachen wie JavaScript ebenfalls unterschieden.

Dringend zu empfehlen ist das Bestimmen der gewünschten Breite und Höhe des eingebetteten Frame-Fensters. Die Breite geben Sie im einleitenden `<iframe>`-Tag mit dem Attribut `width=` an, die Höhe mit `height=`. Erlaubt sind Pixelangaben oder Prozentwerte, die sich auf die Größe des verfügbaren Raums beziehen.

Zwischen dem einleitenden `<iframe>` und dem erforderlichen, abschließenden `</iframe>` können Sie Text und andere Elemente notieren, die von Web-Browsern angezeigt werden, die den eingebetteten Frame nicht interpretieren.

**Beachten Sie:** Wenn Sie das `iframe`-Element in einer HTML-Datei notieren, müssen Sie in dieser Datei die HTML-Variante »Transitional« wählen. Eingebettete Frames gehören also auch nicht zur »reinen Lehre«. Andere Möglichkeiten, um Inhalte aus anderen Dateien innerhalb einer HTML-Datei anzuzeigen, sind Datendateien als Objekt (Kapitel 11.1.2, ebenfalls HTML-Standard), Layer mit Inhalt aus anderer Datei (Kapitel 12.1.1, Netscape-spezifisch) sowie Server Side Includes (HTML-Standard-konform, jedoch nur in HTTP-Umgebung ausführbar).

### 9.4.2 Eigenschaften eingebetteter Frames

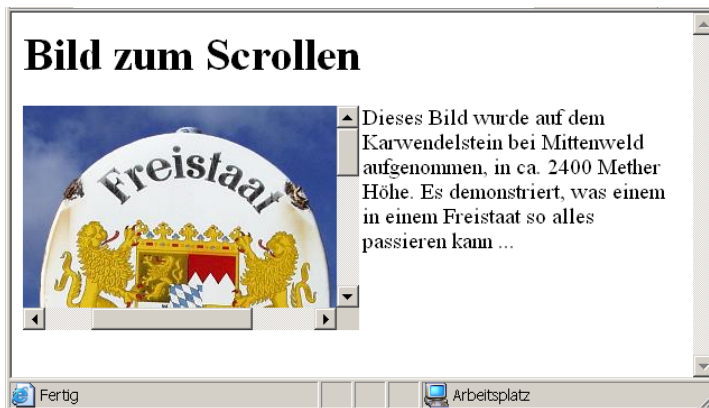
Sie können verschiedene weitere Eigenschaften definieren, um ein eingebettetes Frame-Fenster zur Umgebung auszurichten. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Bild zum Scrollen</h1>

<iframe src="bild.jpg" name="Bildframe" width="300" height="200" align="left"
  scrolling="yes" marginheight="0" marginwidth="0" frameborder="0">
  <p>Ihr Browser kann leider keine eingebetteten Frames anzeigen:
  </p>
</iframe>
  <p>Dieses Bild wurde auf dem Karwendelstein bei Mittenwald
  aufgenommen, in ca. 2400 Meter Höhe. Es demonstriert, was einem in
  einem Freistaat so alles passieren kann ...</p>

</body>
</html>
```



9.13: Über Attribute steuern Sie das Aussehen von eingebetteten Frames.

Das Beispiel zeigt, wie Sie ein eingebettetes Frame-Fenster dazu verwenden können, um eine größere Grafik anzuzeigen und diese mit eigenen Scrollbars auszustatten.

Mit `align="left"` richten Sie den eingebetteten Frame links aus. Nachfolgende Inhalte fließen dann rechts um den Frame. Mit `align="right"` richten Sie den eingebetteten Frame rechts aus und nachfolgende Inhalte fließen links um den Frame. Dieses Attribut ist jedoch als *deprecated* eingestuft und durch die CSS-Eigenschaft `text-align` ersetzbar (z.B. `<iframe style="text-align:left">`).

Mit `scrolling="yes"` können Sie Scrollbars (Bildlaufleisten) in dem eingebetteten Frame erzwingen, mit `scrolling="no"` unterdrücken. `scrolling="auto"` können Sie ebenfalls angeben, doch dies entspricht der Voreinstellung. Mit `frameborder="0"` können Sie den sichtbaren Außenrahmen des eingebetteten Frames unterdrücken. `frameborder="1"` ist die Voreinstellung.

Zwei weitere Attribute, nämlich `hspace=` und `vspace=`, werden vom Internet Explorer ebenfalls interpretiert, sind aber im Zusammenhang mit eingebetteten Frames nicht im HTML-Standard verzeichnet – was etwas inkonsistent ist, da das `align`-Attribut bei solchen Frame-Fenstern genauso wirkt wie etwa bei Grafiken oder Objekten. Mit `hspace= [Pixel]` bestimmen Sie den Abstand zum Text neben dem eingebetteten Frame, mit `vspace= [Pixel]` den Abstand zum Text über bzw. unter dem Frame (*hspace = horizontal space = horizontaler Abstand, vspace = vertical space = vertikaler Abstand*). Beide Angaben erzeugen jedoch ungültiges HTML.

**Beachten Sie:** Genau wie bei normalen Frames besteht auch bei eingebetteten Frames die Möglichkeit, mit dem Attribut `longdesc=` eine Langbeschreibung (Kapitel 9.1.5) zu referenzieren.

### 9.4.3 Verweise zu eingebetteten Frames

In einer HTML-Datei, in der Sie einen eingebetteten Frame definieren, können Sie mit Verweisen auch den Inhalt des eingebetteten Frames ändern. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Fernsteuerung</h1>

<iframe src="startseite.htm" name="Fensterlein"
  width="500" height="400" marginheight="10"
  marginwidth="10" align="right">
<p>Ihr Browser kann leider keine eingebetteten Frames anzeigen</p>
</iframe>
<p>
<a href="willkommen.htm" target="Fensterlein">
  <b>andere Seite</b>
</a><br>
<a href="startseite.htm" target="Fensterlein">
  <b>erste Seite</b>
</a>
</p>

</body>
</html>
```



9.14: Ein Klick auf die Links zeigt im eingebetteten Frame jeweils eine andere Web-Seite an.

Voraussetzung für Verweise zu einem eingebetteten Frame ist, dass der eingebettete Frame mit `name=` einen Namen erhalten hat. Im obigen Beispiel ist das der Name `Fensterlein`. Für das Setzen eines Verweises zu dem eingebetteten Frame gelten die üblichen Regeln zum Referenzieren in HTML. Damit das Verweisziel in dem Frame angezeigt wird, notieren Sie im einleitenden Verweis-Tag zusätzlich die Angabe `target=` (*target* = Ziel). Dahinter folgt der Name des Frame-Fensters, in dem das Verweisziel angezeigt werden soll.

## 9.5 Frames als Mittel für Seitenlayouts

### 9.5.1 Sinnvolle Einsatzmöglichkeiten für Frames

Frames, also die Informationsverteilung auf mehrere, voneinander unabhängige Fenster sind ein faszinierendes Werkzeug, aber auch ein Werkzeug, mit dem man viel verkehrt machen kann. Generell gilt: Jeder Einsatz von Frames muss gerechtfertigt sein. Das bedeutet: Die Verwendung der Frame-Technik muss dem Anwender als sinnvoll und vorteilhaft erscheinen. Wer Frames aus purem Selbstzweck einsetzt, muss damit rechnen, als technikverliebter HTML-Novize statt als souveräner Web-Designer betrachtet zu werden.

Der Grund dafür ist, dass Frames nicht wegzudiskutierende Nachteile haben:

- Probleme bei nicht framefähigen Web-Browsern: Frames werden nicht von allen Web-Browsern angezeigt. Da die gesamte Struktur eines auf Frames basierenden Projekts von der Struktur herkömmlicher Projekte entscheidend abweicht, kann ein Anbieter von Frames solchen Anwendern, die keine Frames anzeigen können, entweder gar keine Alternative anbieten oder eine »zweigleisige« Alternative, die sehr aufwendig zu realisieren und zu pflegen ist.
- Frames und Bildschirmauflösung: Bei kleineren Bildschirmen, z.B. bei Notebooks oder Handheld-Computern, sind mehr als zwei Frame-Fenster bereits eine Zumutung fürs Auge und die Übersicht.
- Ladezeiten: Frames verlangen mehr HTTP-Kommunikation zwischen Browser und Server, da insgesamt mehr Dateien geladen werden müssen. Im Web kann es unter ungünstigen Verhältnissen leichter zu längeren Ladezeiten kommen.
- Problematisches Direktansteuern von untergeordneten Seiten: Es ist zwar theoretisch möglich, aber meistens nicht im Sinne des Anbieters, wenn andere Anwender ein Lesezeichen oder einen Verweis auf eine HTML-Datei setzen, die Teil eines Framesets ist. Das ist in vielen Fällen ärgerlich. So wird beispielsweise anderen Informationsanbietern die Möglichkeit genommen, in einem bestimmten Informationszusammenhang auf eine bestimmte Seite in einem fremden Projekt zu verweisen.

In folgenden Fällen ist der Einsatz von Frames für den Anwender am ehesten nachvollziehbar:

- Zum schnellen Wechseln zwischen Informationseinheiten: In diesem Fall enthält ein Frame-Fenster ein umfangreiches Verzeichnis mit anklickbaren Verweisen auf einzelne Informationsseiten, die in einem anderen, festen Frame-Fenster angezeigt werden. Das »Inhaltsverzeichnis« bleibt also jederzeit eingeblendet, und der Anwender kann zu jedem Zeitpunkt einen neuen Verweis daraus auswählen. Das erspart dem Anwender den wiederholten Rück-

sprung von den einzelnen Informationsseiten auf das übergeordnete Verzeichnis.

*Beispiel:* Ein Anbieter von Ferienappartements kann alle zur Verfügung stehenden Objekte in einem Verweis-Verzeichnis auflisten und das jeweils ausgewählte Objekt in einem festen anderen Frame-Fenster anzeigen.

- Zum ständigen Einblenden projektglobaler Steuerverweise: Bei umfangreichen Projekten, in denen dem Anwender das Gefühl des »lost in hyperspace« droht, ist es sinnvoll, in einem separaten Frame-Fenster immer gültige Steuerverweise anzubieten, z.B. zur Homepage, zur nächsthöheren logischen Ebene, zum Stichwortverzeichnis oder zur Suchdatenbank. Bei kleinen Projekten, die nur aus einer Handvoll Seiten bestehen, wirkt diese Technik dagegen übertrieben und vermittelt dem Anwender eine falsche Vorstellung von der Größe des Projekts. Wenn der Anwender in einem solchen Fall nach wenigen Mausklicks feststellt, dass er bereits alles gesehen hat, wird er um so enttäuschter sein.

*Beispiel:* Eine Zeitung, die im Web ein großes Archiv mit älteren Artikeln anbietet, könnte mit Hilfe der Frame-Technik ständige Verweise zu einem thematisch sortierten Zugangsverzeichnis, zu einem Stichwortverzeichnis und einer Volltextsuchdatenbank für die einzelnen Artikel anbieten.

- Zum gleichzeitigen Anzeigen von zu vergleichenden Informationen: Hypertext bedeutet nicht nur, dem Anwender per Mausklick weitere Informationen zur Verfügung zu stellen, sondern auch, dem Anwender die Möglichkeit zu bieten, sich selbst Informationen so zusammenzustellen, dass er sie optimal miteinander vergleichen und daraus Schlüsse oder Entscheidungen ableiten kann. Zu diesem Zweck eignet sich die Frame-Technik hervorragend, da sie es erlaubt, verschiedene, getrennt voneinander gespeicherte Informationen auf Anwenderwunsch gleichzeitig anzuzeigen.

*Beispiel:* Eine Verbraucherberatung könnte in einem viergeteilten Frameset in zwei Frames zwei gleichartig aufgebaute Verweis-Verzeichnisse zu Produkttests anbieten. Im dritten Frame-Fenster wird der Produkttest angezeigt, den der Anwender im ersten Frame-Fenster mit Verweisen auswählt; im vierten Frame-Fenster kann der Anwender einen Produkttest anzeigen, den er im zweiten Frame-Fenster mit Verweisen auswählt. Auf diese Weise kann der Anwender beliebige getestete Produkte direkt miteinander vergleichen. Voraussetzung hierzu ist natürlich, dass alle Produkttests einen einheitlichen Aufbau und ein einheitliches Bewertungsschema haben, um direkte Vergleiche zu erlauben.

- Bei besonders kunstvoller Seitengestaltung: In diesem Fall muss der Anwender auf den ersten Blick erkennen können, dass die Frame-Technik eine bestimmte künstlerische Aussage unterstützen soll.

Bei Verwendung von Frames besteht auch noch eher als bei »einfachen« Seiten die Gefahr eines gestalterischen Overkills. Achten Sie bei Frames unbedingt darauf, dass die Farben der Inhalte in den verschiedenen Frame-Fenstern insgesamt miteinander harmonisieren. Das Gleiche gilt auch für andere optische Eigenschaften wie Schriftarten, Schriftgrößen usw. Ein Navigationsfenster muss nicht krampfhaft »anders« aussehen als ein inhaltstragendes Fenster, damit auch ja jeder Besucher sieht, dass es sich um Frames handelt. Kontraste zwischen Frame-Fenstern dürfen nicht die Regeln einer homogenen Gesamtgestaltung verletzen.

### 9.5.2 Unfaire Schaufenster-Effekte

Mit Hilfe von Frames ist es auch möglich, fremde Web-Seiten innerhalb eines Frame-Fensters im eigenen Web-Projekt darzustellen. Dies ist jedoch sehr problematisch und in den meisten Fällen unfair. Denn auf diese Weise verkommen fremde Web-Seiten zu »Schaufenstern« innerhalb des eigenen Web-Projekts. Das ist eine unfeine Geste gegenüber den Fremdanbietern, weil es das eigene Web-Projekt in den Augen des Anwenders als »Mega-Projekt« erscheinen lässt. Es ist auch juristisch bedenklich, da Sie sich leicht eine Klage wegen Urheberrechtsverletzung einhandeln können, wenn Sie Fremdinhalte »vereinnahmen«.

Ferner kann es auch passieren, dass in einem Frame-Fenster eine Fremdanbieter-Seite angezeigt wird, die ihrerseits Frames einsetzt. So kommt es zu einem ärgerlichen »Fraktal-Effekt« von Frame-Fenstern auf dem Bildschirm des Anwenders.

Wenn Sie innerhalb eines Frame-Projekts auf fremde Seiten verweisen, sollten Sie eine der folgenden Möglichkeiten wählen:

- Ursprünglichen Fensterzustand wiederherstellen: Setzen Sie den Verweis so, dass der Anwender Ihr definiertes Frameset verlässt und wieder den Fensterzustand erhält, den er vor Aufruf Ihres Projekts hatte. Dies erreichen Sie durch Verweise von der Art:  
`<a href="http://..." target="_parent">Verweistext</a>`
- Doppelverweis anbieten: Dadurch überlassen Sie dem Anwender die Entscheidung, wie er die fremde Seite angezeigt bekommen will. Notieren Sie zwei Verweise direkt unter- oder nebeneinander, die das gleiche Ziel haben, etwa in der Weise:  
`<a href="http://..." target="RechtsUnten">Fremde Seite im Frame rechts unten anzeigen</a><br>  
<a href="http://..." target="_parent">Fremde Seite voll anzeigen</a>`
- Anwender auf Möglichkeit zum Verändern der Frames hinweisen: Wenn Sie keine der beiden anderen Möglichkeiten anbieten wollen, sollten Sie wenigstens das Verändern der Frame-Fenstergrößen durch den Anwender erlauben und den Anwender auf diese Möglichkeit explizit hinweisen. Um das Verändern der Frame-Fenster durch den Anwender zu erlauben, verzichten Sie einfach auf das Attribut `noresize` im `<frame>`-Tag.

Mit Hilfe von JavaScript ist es möglich, die Seitenanzeige in Frames zu verhindern (Kapitel 19.2).

# 10 Formulare

## 10.1 Formulare definieren

### 10.1.1 Allgemeines zu Formularen

HTML stellt die Möglichkeit zur Verfügung, Formulare zu erstellen. In Formularen kann der Anwender Eingabefelder ausfüllen, in mehrzeiligen Textfeldern Text eingeben, aus Listen Einträge auswählen usw. Wenn das Formular fertig ausgefüllt ist, kann der Anwender auf einen Button klicken, um das Formular abzusenden.

Dazu geben Sie beim Erstellen eines Formulars an, was mit den Daten des ausgefüllten Formulars passieren soll. Sie können sich die ausgefüllten Daten beispielsweise per E-Mail zuschicken lassen oder von einem CGI-Programm auf dem Server-Rechner weiterverarbeiten lassen. Formulare können sehr unterschiedliche Aufgaben haben. So werden sie zum Beispiel eingesetzt,

- um bestimmte, gleichartig strukturierte Auskünfte von Anwendern einzuholen,
- um Anwendern das Suchen in Datenbeständen zu ermöglichen,
- um Anwendern die Möglichkeit zu geben, selbst Daten für einen Datenbestand beizusteuern,
- um dem Anwender die Möglichkeit individueller Interaktion zu bieten, etwa um aus einer Produktpalette etwas Bestimmtes zu bestellen.

Ein Softwarehersteller könnte z.B. ein Formular zur Verfügung stellen, in dem der Anwender angeben kann, welche Produkte der Firma er besitzt, wie er Kenntnis von den Produkten erhalten hat, welchen Beruf er ausübt, auf welchem Rechnertyp die Software bei ihm läuft usw. Auf diese Weise könnte er gleichartig strukturiertes und daher gut vergleichbares Feedback von Anwendern einholen.

Viele Suchdienste im Internet bieten dem aufrufenden Web-Browser Eingabeformulare an, in denen der Anwender seinen Suchwunsch spezifizieren kann. Ohne solche Formulare wäre das Durchsuchen gar nicht möglich. Die meisten Suchdienste bieten darüber hinaus auch die Möglichkeit an, eigene Internet-Adressen in die Datenbank einzuspeisen. Das Einholen der dazu nötigen Information geschieht ebenfalls mit Hilfe von Formularen.

Immer zahlreicher werden auch die Online-Shops im Internet. Egal ob Tickets, Pizza oder Unterwäsche – um solche Bestell-Services zu realisieren, sind Formulare erforderlich, in denen der Anwender seine Bestellwünsche genau angeben kann.

Wenn Sie dem Anwender auf Ihren Web-Seiten einfach nur eine Möglichkeit anbieten wollen, direkt mit Ihnen in Kontakt zu treten, genügt allerdings auch ein einfacher E-Mail-Verweis an die eigene E-Mail-Adresse (Kapitel 6.4).

**Beachten Sie:** Nicht wenige Anbieter versuchen über Formulare, Anwenderadressen zu sammeln, um diese dann zu verkaufen oder um Auskünfte einzuholen, zu denen ein Anwender nicht verpflichtet ist. Dazu wird häufig ein Anwenderbegehren als Köder benutzt – beispielsweise ein Gewinnspiel oder die Download-Möglichkeit der kostenlosen Ausprobierversion einer teuren Software.

Dieser Missbrauch hat einige selbsternannte Hüter des Datenschutzes auf den Plan gerufen, die aber leider an der falschen Ecke kehren und sich nicht an die rechtlich meist gut abgesicherten Datenjäger wenden, sondern an ahnungslose Homepage-Besitzer, die ein kleines Formular im Web haben und plötzlich dafür abgemahnt werden. Dabei wird mit dem Prinzip der so genannten »Datensparsamkeit« argumentiert (siehe auch Hinweise zum Teledienste-Datenschutzgesetz (TDDSG) im Abschnitt *Gesetze für »neue Medien« (in Deutschland)*, Kapitel 25.6.4). Versuchen Sie also nicht, den Anwender mit Hilfe von Formularen wie eine Zitrone auszuquetschen. Beschränken Sie sich auf solche Formularfelder, die für den Formularzweck erkennbar nötig sind.

### 10.1.2 Formularbereich definieren

Sie können an einer beliebigen Stelle innerhalb des Dateikörpers einer HTML-Datei ein Formular definieren.

Beispiel 1:

```
<form action="http://www.ihr-guter-name.de/cgi-bin/feedback.pl" method="get">
<!-- hier folgen die Formularelemente -->
</form>
```

Beispiel 2:

```
<form action="mailto:eierschale@irgend.wo" method="post" enctype="text/plain">
<!-- hier folgen die Formularelemente -->
</form>
```

Mit `<form>...</form>` definieren Sie ein Formular (*form* = *Formular*). Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden Tag `</form>` steht, gehört zum Formular. Das sind hauptsächlich Elemente des Formulars wie Eingabefelder, Auswahllisten oder Buttons. Um die Elemente zu platzieren und zu beschriften, können Sie dazwischen aber auch andere HTML-Elemente notieren. Dabei müssen Sie allerdings Folgendes beachten: Nach der HTML-Variante »Strict« dürfen Sie innerhalb eines Formulars nur Block-Elemente (und Script-Bereiche) notieren, also etwa Überschriften, Textabsätze, allgemeine Bereiche oder Tabellen. In der HTML-Variante »Transitional« ist es dagegen außerdem erlaubt, zwischen `<form>` und `</form>` auch gemischten Inhalt aus Text und Inline-Elementen zu notieren. Um die etwas lästige und an dieser Stelle nicht ganz nachvollziehbare Restriktion in der »Strict«-Variante zu umgehen, können Sie sich damit behelfen, dass Sie das Formular folgendermaßen strukturieren:

```
<form><div> <!-- Formularinhalt --> </div></form>
```

Innerhalb des `div`-Elements, das da innerhalb des Formulars notiert ist, haben Sie auch in der »Strict«-Variante die Freiheiten der »Transitional«-Variante. Eine andere Möglichkeit besteht darin, anstelle des `div`-Elements mit dem `fieldset`-Element zu arbeiten und Formularinhalte



darin einzuschließen – auch innerhalb des `fieldset`-Elements haben Sie alle Freiheiten (siehe dazu Abschnitt 10.8, *Elemente gruppieren*).

Im einleitenden `<form>`-Tag geben Sie mit dem Pflichtattribut `action=` an, was mit den Formulardaten passieren soll, wenn der Anwender das Formular absendet (*action = Aktion*). Die Wertzuweisung an `action=` kann beispielsweise eine E-Mail-Adresse (normalerweise Ihre eigene) mit vorangestelltem `mailto:` sein – so wie im zweiten der obigen Beispiele (*mailto = sende an*). Dann werden die ausgefüllten Formulardaten an diese E-Mail-Adresse geschickt, sofern das möglich ist. Oder Sie rufen ein Programm auf dem Server-Rechner auf, meistens ein CGI-Script, das die Daten weiterverarbeitet – so wie im ersten der obigen Beispiele.

Sie können bei `action=` auch eine HTML-Datei angeben. Diese wird bei Absenden des Formulars aufgerufen und kann die Formulardaten z.B. mit JavaScript weiterverarbeiten. Das ist beispielsweise für mehrseitige Formulare interessant. Berücksichtigen Sie dabei aber, dass JavaScript nur dann Zugriff auf Daten hat, wenn die Methode `get` verwendet wurde. Bei einigen Browsern, z.B. bei Opera, funktioniert das Übergeben von Formulardaten zwischen HTML-Dateien auch nur in HTTP-Umgebung, also nicht lokal ohne Server-Kommunikation.

Bei der Wertzuweisung an `action=` gelten die Regeln zum Referenzieren in HTML. Das Referenzieren mit relativen oder absoluten Pfadangaben ist also ebenso möglich wie das oben in Beispiel 1 gezeigte Referenzieren mit einem absoluten URI, z.B.:

```
<form action="/cgi-bin/auswert.pl">
<form action="../../../cgi-bin/suche.cgi">
```

Eine weiteres wichtiges Attribut bei der Formulardefinition ist das Attribut `method=`. Dabei bestimmen Sie, nach welcher HTTP-Übertragungsmethode die Formulardaten an ihr Ziel gelangen. Dabei gibt es zwei mögliche Wertzuweisungen:

- Wenn Sie `method="get"` wählen (diese Angabe ist übrigens nicht zwingend erforderlich, da `get` als Default-Einstellung definiert ist), werden die Daten des ausgefüllten Formulars als Parameter an die Aufrufadresse angehängt. Die Anfrage, die beim Server eintrifft, sieht dann beispielsweise so aus: `http://www.ihr-guter-name.de/cgi-bin/feedback.cgi?AnwenderName=Stefan+M%FCnz&AnwenderMail=selhtml@teamone.de&Text=Das+ist+e+in+kleiner+Text`. Das verarbeitende Script kann diese als Parameter übergebene Zeichenkette auslesen und für die Datenverarbeitung auseinander dröseln.
- Wenn Sie `method="post"` wählen, werden die Daten des ausgefüllten Formulars vom Web-Server über den Standardeingabekanal zur Verfügung gestellt, und ein auswertendes CGI-Script muss die ankommenden Daten behandeln wie eine Benutzereingabe, die auf der Kommandozeile gemacht wurde (*post = verschicken*). Bei CGI-Scripts müssen Sie diese Methode auf jeden Fall dann verwenden, wenn das Script die POST-Methode verarbeitet und wenn die Formulardaten zu umfangreich für die GET-Methode sind.

Das W3-Konsortium empfiehlt, die Methode `get` dann zu wählen, wenn das auswertende Programm die Daten nur als Input benötigt (z.B. für eine Suche), während die Methode `post` für Fälle empfohlen wird, in denen das auswertende Programm »Seiteneffekte« wie das Einspeisen der Daten in eine Datenbank ausführt.

Wenn Sie sich die Formulardaten per E-Mail zuschicken lassen, wie im zweiten obigen Beispiel gezeigt, dann benutzen Sie immer `method="post"`. Außerdem sollten Sie bei E-Mail-Empfang von Formulardaten im einleitenden `<form>`-Tag `enctype="text/plain"` mit angeben. Denn

Formulardaten sind normalerweise nach einem Schema kodiert, das für Menschen keine Freude zu lesen ist. Mit der genannten Angabe erhalten Sie zumindest von Anwendern, die Ihr Formular mit einem modernen Web-Browser ausfüllen, ordentlich formatierte E-Mails.

**Beachten Sie:** Bei E-Mail-Formularen besteht keine Garantie auf Erfolg. Es hängt vom Browser und anderen Einstellungen auf dem Rechner des Anwenders ab, ob der Formularversand klappt. E-Mail-Formulare gelten deshalb mittlerweile als unsauber, zumal es Alternativen gibt. Mehr dazu im Abschnitt 10.12 *Formulare verarbeiten*.

### 10.1.3 Zielfenster für Server-Antwort

Wenn Sie mit Frames arbeiten und in einem Frame-Fenster ein Formular haben, nach dessen Absenden die Server-Antwort bzw. die Antwort eines CGI-Scripts in einem anderen Frame-Fenster angezeigt werden soll, können Sie das gewünschte Zielfenster angeben. Ein Beispiel:

```
<form action="/cgi-bin/auswert.pl" method="get" target="Daten">
<!-- hier folgen die Formularelemente -->
</form>
```

Mit dem Attribut `target=` können Sie im einleitenden `<form>`-Tag den Namen des Frame-Fensters angeben, in dem die Server-Antwort ausgegeben werden soll. Es muss sich entweder um einen Fensternamen handeln, der für ein Frame-Fenster dem `name`-Attribut im `<frame>`-Tag vergeben wurde, oder um einen der folgenden reservierten Fensternamen:

- `_self`, um die Server-Antwort im aktuellen Fenster auszugeben,
- `_parent`, um für die Server-Antwort bei verschachtelten Framesets das aktuelle Frameset zu sprengen,
- `_top`, um für die Server-Antwort bei verschachtelten Framesets alle Framesets zu sprengen.

**Beachten Sie:** Das `target`-Attribut ist zwar nicht als *deprecated* gekennzeichnet, doch um es einzusetzen, müssen Sie die HTML-Variante »Transitional« verwenden. Der Grund ist, dass dieses Attribut vorwiegend für Verweise bei Verwendung von Frames gedacht ist und Frames eine eigene HTML-Variante haben, die von der Einstufung her der Variante »Transitional« entspricht (auf gut Deutsch: nicht der »reinen Lehre« entspricht).

### 10.1.4 Zu verarbeitende Zeichensätze

Sie können angeben, nach welchen Zeichensätzen eingegebene Formulardaten interpretiert werden sollen. Dabei können Sie einen oder mehrere Zeichensätze angeben. Ein Beispiel:

```
<form action="/cgi-bin/auswert.pl" method="get" accept-charset="ISO-8859-1, ISO-
8859-2">
<!-- hier folgen die Formularelemente -->
</form>
```

Mit dem Attribut `accept-charset=` können Sie Zeichensätze angeben. Trennen Sie mehrere Angaben durch Kommata und/oder Leerzeichen. Erlaubt sind Zeichensatzangaben, wie sie auf der Web-Adresse <http://www.iana.org/assignments/character-sets> angegeben sind.

### 10.1.5 Weitere Informationen

Zur Datenverarbeitung von Formularen mit Hilfe von CGI-Scripts finden Sie zusätzliche Informationen im Kapitel 10.12.

## 10.2 Eingabefelder und Eingabebereiche

### 10.2.1 Einzeilige Eingabefelder definieren

Einzeilige Eingabefelder dienen zur Aufnahme von einem oder wenigen Wörtern oder einer Zahl. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Formular f&uuml;r Namenseingabe</h1>

<form action="input_text.htm">
<p>Vorname:<br>
<input name="vorname" type="text" size="30" maxlength="30">
</p>
<p>Zuname:<br>
<input name="zuname" type="text" size="30" maxlength="40">
</p>
</form>

</body>
</html>
```

`<input>` definiert ein einzeiliges Eingabefeld (*input* = *Eingabe*). Der Vollständigkeit halber sollten Sie die Angabe `type="text"` dazusetzen.

Jedes Eingabefeld sollte einen internen Bezeichnernamen erhalten, und zwar mit der Angabe `name=`. Der vergebene Name wird bei CGI-Scripts benötigt, um auf die Daten des Eingabefeldes zugreifen zu können. Bei E-Mail-Formularen taucht der Name als Feldname auf. Und auch bei dem Formularfeldzugriff mit JavaScript ist der Name von Bedeutung. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (`_`), den Bindestrich (`-`), den Doppelpunkt (`:`) oder den Punkt (`.`). Im Hinblick auf JavaScript darf der Name sogar nur Buchstaben, Ziffern und den Unterstrich (`_`) enthalten. Groß- und Kleinschreibung wird bei den meisten Programmiersprachen ebenfalls unterschieden.

10.1: Zwei einzeilige Eingabefelder mit vorgeschriebener Größe.

Ferner sollten Sie bei einzeiligen Eingabefeldern immer die Anzeigelänge in Zeichen mit `size=` sowie die interne Feldlänge in Zeichen `maxlength=` bestimmen. Beide Angaben bedeuten die Anzahl Zeichen (*size = Größe, maxlength = maximal length = maximale Länge*). Wenn die interne Feldlänge `maxlength=` größer ist als die angezeigte Feldlänge `size=` (wie im zweiten Eingabefeld des obigen Beispiels), dann wird bei längeren Eingaben automatisch gescrollt (im Beispiel also ab dem 31. eingegebenen Zeichen).

Um Beschriftung und Eingabefelder ordentlich zu positionieren, empfiehlt sich übrigens der Einsatz einer »blinden« Tabelle (Kapitel 8.6). Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Formular für Namenseingabe</h1>

<form action="input_text_tabelle.htm">
<table border="0" cellpadding="0" cellspacing="4">
  <tr>
    <td align="right">Vorname:</td>
    <td><input name="vorname" type="text" size="30" maxlength="30"></td>
  </tr><tr>
    <td align="right">Zuname:</td>
    <td><input name="zuname" type="text" size="30" maxlength="40"></td>
  </tr>
</table>
</form>

</body>
</html>
```

10.2: Dieselben Eingabefelder wie in Bild 10.1, diesmal mit einer »blinden« Tabelle und Beschriftung sauber ausgerichtet.

Das Beispiel zeigt das gleiche Formular wie im Beispiel weiter oben. Diesmal stehen jedoch Beschriftung und zugehöriges Formularfeld in je einer Tabellenzeile sauber nebeneinander. Die Beschriftungstexte werden dabei rechtsbündig ausgerichtet und erscheinen dadurch bündig zu den Eingabefeldern hin orientiert. Wichtig ist, dass das `form`-Element außerhalb der Tabelle steht, oder andersherum, dass die Tabelle als Element innerhalb des Formulars notiert wird.

**Beachten Sie:** Das `<input>`-Tag ist ein so genanntes Standalone-Tag. Wenn Sie XHTML-konform arbeiten, müssen Sie das `input`-Element als inhaltsleer kennzeichnen. Dazu notieren Sie das allein stehende Tag in der Form `<input... />`. Weitere Informationen dazu im Kapitel 14.1.11 zu XHTML und HTML.

Mit Hilfe von JavaScript können Sie die Eingaben von Anwendern vor dem Absenden des Formulars kontrollieren. So können Sie zum Beispiel erzwingen, dass in einem Eingabefeld eine Zahl in einem bestimmten Wertebereich eingegeben wird usw. In Kapitel 19.6 finden Sie ein Anwendungsbeispiel zum Thema *Formulareingaben überprüfen*.

### 10.2.2 Textvorbelegung bei einzeiligen Eingabefeldern

Sie können ein einzeiliges Eingabefeld mit einem Inhalt vorbelegen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Einfach &uuml;berschreiben...</h1>

<form action="input_text.htm">
<p>Vorname:<br>
<input name="vorname" type="text" size="30" maxlength="30" value="Michaela">
</p>
<p>Zuname:<br>
```

```
<input name="zuname" type="text" size="30" maxlength="40" value="Mustermann">
</p>
</form>

</body>
</html>
```

Mit dem zusätzlichen Attribut `value=` können Sie einen Text bestimmen, mit dem das Feld vorgelegt wird (*value = Wert*).

### 10.2.3 Eingabefelder für Passwörter

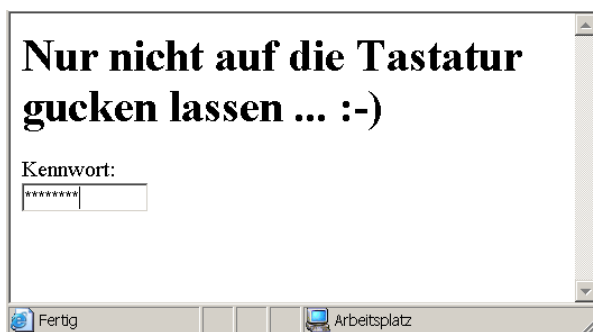
Für die Eingabe von Geheimnummern, Passwörtern usw. gibt es einen speziellen Typ von Eingabefeld. Die eingegebenen Zeichen werden dabei durch Platzhalter (meistens Sternchen) dargestellt, sodass Personen im Raum des Anwenders nicht zufällig das eingegebene Passwort mitlesen können. Eingabefelder für Passwörter werden mit `type="password"` im `<input>`-Tag definiert. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Nur nicht auf die Tastatur gucken lassen ... :-)</h1>

<form action="input_password.htm">
<p>Kennwort:<br>
<input name="kennwort" type="password" size="12" maxlength="12">
</p>
</form>

</body>
</html>
```



10.3: Eingabefelder für Passwörter zeigen nur Sternchen statt der eingetippten Zeichen an.

**Beachten Sie:** Passwörter werden beim normalen HTTP-Protokoll trotz der verdeckten Eingabe im Klartext über das Internet übertragen. Weisen Sie Anwender in ernsthaften Zusammenhängen auf diese Tatsache bitte explizit hin. Für eine verschlüsselte Kommunikation zwischen Browser und Server gibt es das HTTPS-Protokoll, das der Web-Server unterstützen muss. Fragen Sie gegebenenfalls Ihren Provider oder Webmaster danach. Ein Passwortfeld in HTML ist keine Abfrage für geschützte Web-Seiten. Lesen Sie zu diesem Thema den Anhang A.6 über das Thema *.htaccess – Server-Reaktionen kontrollieren*.

### 10.2.4 Mehrzeilige Eingabebereiche definieren

Mehrzeilige Eingabefelder dienen zur Aufnahme von Kommentaren, Nachrichten usw. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ganz spontan</h1>

<form action="textarea.htm">
<p>Welche HTML-Elemente fallen Ihnen ein,
    und was bewirken sie:<br>
<textarea name="user_eingabe" cols="50" rows="10"></textarea>
</p>
</form>

</body>
</html>
```

`<textarea ...>` leitet ein mehrzeiliges Eingabefeld ein (*textarea* = *Textbereich*). Jedes mehrzeilige Eingabefeld sollte ebenso wie ein einzeliliges Eingabefeld einen internen Bezeichnernamen erhalten, und zwar mit der Angabe `name=`.

Pflicht ist die Angabe zur Höhe und Breite des Eingabebereichs. Das Attribut `rows=` bestimmt die Anzahl der angezeigten Zeilen (*rows* = *Zeilen*) und damit die Höhe, während `cols=` die Anzahl der angezeigten Spalten (*cols* = *columns* = *Spalten*) festlegt. »Spalten« bedeutet dabei die Anzahl Zeichen (pro Zeile). Mit `</textarea>` schließen Sie das mehrzeilige Eingabefeld ab. Das End-Tag ist nötig und darf nicht weggelassen werden.

10.4: Ein mehrzeiliges Eingabefeld mit 50 Zeichen Breite und zehn Zeichen Höhe.

**Beachten Sie:** Die Attribute `rows=` und `cols=` bestimmen lediglich die Anzeigegröße des Eingabebereichs, nicht die Länge des erlaubten Textes. Die ist theoretisch unbegrenzt. Web-Browser stellen die mehrzeiligen Eingabefelder bei der Anzeige üblicherweise mit Scrollbalken aus, sodass der Anwender bei längeren Eingaben scrollen kann.

### 10.2.5 Textvorbelegung bei mehrzeiligen Eingabebereichen

Sie können ein mehrzeiliges Eingabefeld mit Inhalt vorbelegen. Um mehrzeilige Eingabefelder mit Text vorzubelegen, notieren Sie den gewünschten Text einfach als Elementinhalt zwischen dem einleitenden `<textarea>`-Tag und dem abschließenden `</textarea>`. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

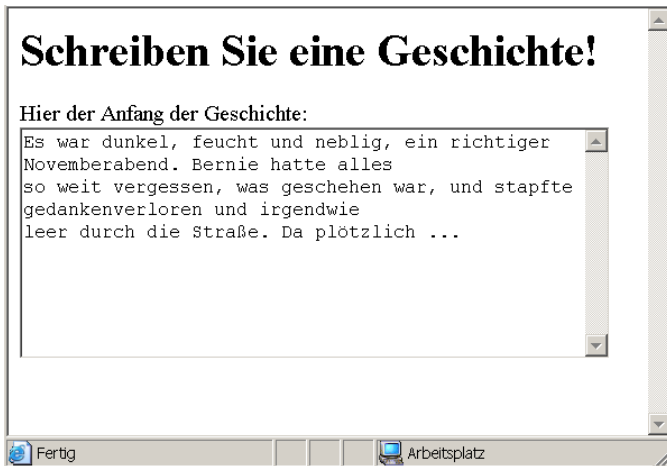
<h1>Schreiben Sie eine Geschichte!</h1>

<form action="textarea.htm">
<p>Hier der Anfang der Geschichte:<br>
<textarea name="user_eingabe" cols="50" rows="10">
Es war dunkel, feucht und neblig, ein richtiger Novemberabend. Bernie hatte alles
so weit vergessen, was geschehen war, und stapfte gedankenverloren und irgendwie
leer durch die Stra&szlig;e. Da pl&ouml;tzlich ...
</textarea></p>
</form>

```



```
</body>
</html>
```



10.5: Um es den Surfern leicht zu machen, geben Sie dem Eingabefeld gleich einen Textvorschlag mit.

### 10.2.6 Zeilenumbruch bei mehrzeiligen Eingabebereichen

Dies gehört nicht zum HTML-Standard, weil das W3-Konsortium wohl davon ausgeht, dass Text innerhalb eines mehrzeiligen Eingabebereichs bei der Eingabe automatisch umbrochen wird. Nicht so jedoch bei Netscape bis einschließlich Version 4.x. Dort wird automatisch quer gescrollt, solange der Anwender keine Return-Taste drückt, um eine neue Zeile zu beginnen. Um die Option des automatischen Zeilenumbruchs zu erlauben, hat Netscape seinerzeit ein Attribut eingeführt, um den Zeilenumbruch kontrollierbar zu machen. Der Internet Explorer interpretiert dieses Attribut ebenfalls. Aber wie es bei proprietären Sprachbestandteilen so ist – mittlerweile ist ein Wildwuchs an Wertzuweisungen entstanden. Und wenn Sie das Attribut verwenden, erzeugen Sie in jedem Fall ungültiges HTML. Ein Beispiel:

```
<html>
<head>
<title>Textumbruch in mehrzeiligen Eingabebereichen</title>
</head>
<body>

<h1>Brechen Sie um, aber nicht ab!</h1>

<form action="textarea_wrap.htm">
<p>einmal soft:<br><textarea name="soft" cols="30" rows="3"
wrap="soft"></textarea></p>
<p>einmal hard:<br><textarea name="hard" cols="30" rows="3"
wrap="hard"></textarea></p>
```

```

<p>einmal physical:<br><textarea name="physical" cols="30" rows="3"
wrap="physical"></textarea></p>
<p>einmal virtual:<br><textarea name="virtual" cols="30" rows="3"
wrap="virtual"></textarea></p>
<p>und einmal off:<br><textarea name="off" cols="30" rows="3"
wrap="off"></textarea></p>
</form>
</body>
</html>

```

10.6: Der Zeilenumbruch in mehrzeiligen Eingabefeldern lässt sich beeinflussen.

Mit dem nichtstandardisierten Attribut `wrap=` können Sie den Zeilenumbruch steuern (*wrap = Umbruch*). Die folgenden Angaben sind »ohne Gewähr«:

- `wrap="soft"` wird von Netscape 4.x interpretiert und bewirkt automatischen Zeilenumbruch bei der Anzeige. Beim Absenden des Formulars werden an Stellen, wo der Text umbrochen ist, jedoch keine Zeilenumbruchzeichen übertragen.

- `wrap="hard"` wird von Netscape 4.x interpretiert und bewirkt automatischen Zeilenumbruch bei der Anzeige. Beim Absenden des Formulars werden die Zeilenumbruchzeichen an Stellen, wo der Text automatisch umbrochen ist, übertragen.
- `wrap="virtual"` wird vom MS Internet Explorer ab Version 4.x und den Netscape-Versionen 2.x und 3.x interpretiert und ist gleichbedeutend mit `wrap="soft"`.
- `wrap="physical"` wird vom MS Internet Explorer ab Version 4.x und den Netscape-Versionen 2.x und 3.x interpretiert und ist gleichbedeutend mit `wrap="hard"`.
- `wrap="off"` wird vom MS Internet Explorer ab Version 4.x und von Netscape interpretiert und ist gleichbedeutend mit der Voreinstellung von Netscape bis einschließlich Version 4.x – nämlich keinen Text automatisch zu umbrechen.

### 10.2.7 Eingabefelder und Eingabebereiche auf »nur lesen« setzen

Sie können erzwingen, dass ein Eingabefeld kein Eingabefeld mehr ist, sondern eigentlich nur ein Ausgabefeld, und ein Eingabebereich nur ein Ausgabebereich. Das kann beispielsweise interessant sein, wenn Sie JavaScript-ermittelte Werte in einem Formularfeld ausgeben, oder Felder mit einem Wert vorbelegen möchten, den der Anwender aber nicht ändern können soll. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Kein Rabatt und keine Gnade</h1>

<form action="input_text_value.htm">
<p>Preis:<br>
<input name="vorname" type="text" size="8" value="&euro; 699.-" readonly>
</p>
<p>Lizenzvereinbarung:<br>
<textarea name="lizenz" cols="50" rows="10" readonly>
Mit dem Absenden des Formulars stimme ich zu, dass ich diese
Lizenzvereinbarung gelesen habe usw.
</textarea>
</p>
</form>

</body>
</html>
```

**Kein Rabatt und keine Gnade**

Preis:

Lizenzvereinbarung:

Fertig Arbeitsplatz

10.7: Die gesperrten Eingabefelder sind grau dargestellt (beim Rollbalken sichtbar).

Mit dem Attribut `readonly` können Sie ein einzeliges Eingabefeld oder einen mehrzeiligen Eingabereich auf »nur lesen« setzen.

**Beachten Sie:** Bei älteren Browsern funktioniert diese Angabe nicht, und der Feldeintrag ist für den Anwender überschreibbar! Wenn Sie **XHTML-konform** arbeiten wollen, müssen Sie das Attribut in der Form `readonly="readonly"` notieren. Weitere Informationen dazu im Kapitel 14.1.11 zu *XHTML und HTML*.

## 10.3 Auswahllisten

### 10.3.1 Auswahllisten definieren

Sie können dem Anwender eine Liste mit festen Einträgen anbieten, aus der er einen Eintrag auswählen kann. Der Text des ausgewählten Eintrags wird übertragen, wenn der Anwender das Formular abschickt. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>W&auml;hlen Sie Ihren Favoriten!</h1>

<form action="select.htm">
<p>
```

```
<select name="top5" size="3">
<option>Heino</option>
<option>Michael Jackson</option>
<option>Tom Waits</option>
<option>Nina Hagen</option>
<option>Marianne Rosenberg</option>
</select>
</p>
</form>

</body>
</html>
```



10.8: Auswahllisten bieten mehrere Optionen übersichtlich und platzsparend an.

`<select ...>` leitet eine Auswahlliste ein. Jede Auswahlliste sollte einen internen Bezeichnernamen erhalten, und zwar mit dem Attribut `name=`. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Ansonsten gelten die in Kapitel 10.2.1 genannten Konventionen für die Namensgebung.

Mit dem Attribut `size=` bestimmen Sie die Anzeigegröße der Liste, d.h., wie viele Einträge angezeigt werden sollen. Wenn die Liste mehr Einträge enthält, als angezeigt werden, kann der Anwender in der Liste scrollen. Wenn Sie `size="1"` angeben, definieren Sie eine so genannte »Dropdown-Liste«.

Mit `<option>...</option>` definieren Sie zwischen dem einleitenden `<select>`-Tag und dem Abschluss-Tag `</select>` jeweils einen Eintrag der Auswahlliste. Hinter `<option>` muss der Text des Listeneintrags stehen. Sie können so viele Listeneinträge definieren wie Sie wollen. Ein Abschluss-Tag `</option>` ist zwar optional, im Hinblick auf verarbeitende Programmiersprachen aber dringend zu empfehlen.

**Beachten Sie:** Die Breite der Listenanzeige wird automatisch ermittelt, abhängig vom längsten Eintrag (es sei denn, Sie wenden das Formatieren mit CSS an). Mit Hilfe von JavaScript können Sie Auswahllisten auch »zweckentfremden«, um Verweise innerhalb Ihres Projekts zu realisieren. In Kapitel 19.9 finden Sie ein Beispiel dazu: *Verweisliste als Auswahlliste*.

### 10.3.2 Auswahllisten mit Mehrfachauswahl

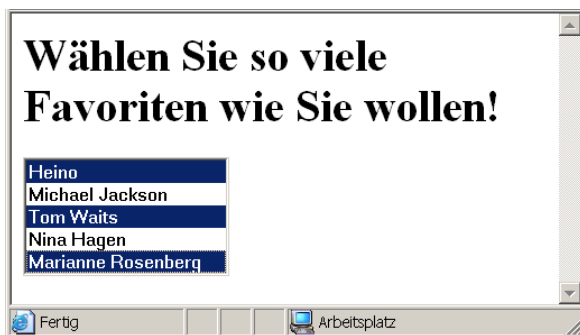
Wenn Sie nichts anderes angeben, kann der Anwender aus einer Auswahlliste genau einen Eintrag auswählen. Sie können auch eine Mehrfachauswahl erlauben. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Wählen Sie so viele Favoriten wie Sie wollen!</h1>

<form action="select.htm">
<p>
<select name="top5" size="5" multiple>
<option>Heino</option>
<option>Michael Jackson</option>
<option>Tom Waits</option>
<option>Nina Hagen</option>
<option>Marianne Rosenberg</option>
</select>
</p>
</form>

</body>
</html>
```



10.9: Die Mehrfachauswahl erlaubt das Selektieren beliebig vieler Listeneinträge (mit Mausklick plus  oder ).

Die Mehrfachauswahl einer Auswahlliste erlauben Sie durch das zusätzliche Attribut `multiple` im einleitenden `<select>`-Tag.

**Beachten Sie:** Eine Mehrfachauswahl ist für Anwender nicht unmittelbar erkennbar. Deshalb sollten Sie darauf hinweisen, wenn mehrere Einträge auswählbar sind. Auch ist nicht allen Anwendern klar, wie sie mehrere Einträge selektieren können. Auf modernen PC-Tastaturen geschieht das normalerweise durch Halten der `[Strg]`-Taste bei gleichzeitigem Anklicken der gewünschten Listeneinträge. Mehrere Einträge markieren Sie mit gedrückter `[⇧]`-Taste, indem Sie zunächst den obersten und dann den untersten gewünschten Eintrag anklicken.

Wenn Sie XHTML-konform arbeiten wollen, müssen Sie das Attribut in der Form `multiple="multiple"` notieren. Weitere Informationen dazu im Kapitel 14.1.11 zu *XHTML und HTML*.

### 10.3.3 Einträge vorselektieren

Wenn Sie nichts anderes angeben, ist zunächst kein Eintrag einer Auswahlliste vorselektiert. Sie können einen Eintrag vorselektieren. In Verbindung mit der Mehrfachauswahl können Sie auch mehrere Einträge vorselektieren. Vorselektierte Einträge haben einen sichtbaren Markierungsbalken. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Sie können auch einen anderen Favoriten wählen!</h1>

<form action="select.htm">
<p>
<select name="top5" size="5">
<option>Heino</option>
<option>Michael Jackson</option>
<option selected>Tom Waits</option>
<option>Nina Hagen</option>
<option>Marianne Rosenberg</option>
</select>
</p>
</form>

</body>
</html>
```

Um einen Eintrag der Auswahlliste vorzuselektieren, geben Sie im einleitenden `<option>`-Tag des betreffenden Eintrags das Attribut `selected` an.

**Beachten Sie:** Wenn Sie XHTML-konform arbeiten wollen, müssen Sie das Attribut in der Form `selected="selected"` notieren. Weitere Informationen dazu im Kapitel 14.1.11 zu *XHTML und HTML*.

### 10.3.4 Absendewert von Einträgen bestimmen

Normalerweise wird beim Absenden des Formulars der Text eines ausgewählten Listeneintrags übertragen, der zwischen `<option>` und `</option>` notiert ist. Sie können jedoch bestimmen, dass intern ein anderer Text übertragen wird. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Pizzeria Fantasia</h1>

<form action="select.htm">
<p>Ihre Pizza-Bestellung:</p>
<p>
<select name="Pizza" size="5"
onchange="alert(this.form.Pizza.options[this.form.Pizza.selectedIndex].value)">
<option value="P101">Pizza Napoli</option>
<option value="P102">Pizza Funghi</option>
<option value="P103">Pizza Mare</option>
<option value="P104">Pizza Tonno</option>
<option value="P105">Pizza Mexicana</option>
<option value="P106">Pizza Regina</option>
<option value="P107">Pizza de la Casa</option>
<option value="P108">Pizza Calzone</option>
<option value="P109">Pizza con tutti</option>
</select>
</p>
</form>

</body>
</html>
```



10.10: Beim Anklicken eines Eintrags zeigt ein Event-Handler den Absendewert des Eintrags an.

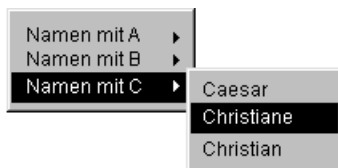


Um für einen Eintrag der Auswahlliste einen anderen Absendewert zu bestimmen, geben Sie im einleitenden `<option>`-Tag des betreffenden Eintrags das Attribut `value=` an (*value = Wert*). Als Wert weisen Sie den gewünschten Absendewert zu. Beim Absenden des Formulars wird dann der hier bestimmte Text eines ausgewählten Eintrags übertragen, nicht der Text, der dem Anwender beim Listeneintrag angeboten wurde.

Im obigen Beispiel ist im einleitenden `<select>`-Tag ein so genannter Event-Handler notiert (Kapitel 18.10). Diesem wird im Beispiel eine JavaScript-Anweisung zugewiesen, die bewirkt, dass ein Meldungsfenster den internen Absendewert des Eintrags zur Kontrolle ausgibt, sobald der Anwender einen Eintrag selektiert.

### 10.3.5 Verschachtelte Auswahllisten definieren

Eine verschachtelte Auswahlliste sollte ein Web-Browser nicht wie üblich als einfache Auswahlliste anzeigen, sondern als verschachtelte Menüstruktur. Anwender moderner grafischer Oberflächen kennen solche verschachtelten Menüs. Das Startmenü von Windows 95 ist beispielsweise ein solches Menü. Netscape 6.0 und Internet Explorer 6.0 erkennen als erste Browser diese HTML-Elemente, stellen sie aber nicht sehr überzeugend dar.



10.11: So sollten die verschachtelten Auswahllisten in etwa aussehen.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>W&auml;hlen Sie Ihren Lieblingsnamen!</h1>

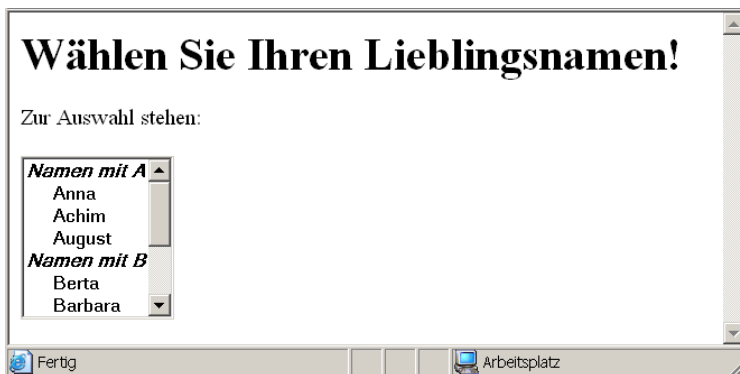
<form action="select_optgroup.htm">
<p>Zur Auswahl stehen:</p>
<p>
<select name="Namen" size=3>
  <optgroup label="Namen mit A">
    <option label="Anna">Anna</option>
    <option label="Achim">Achim</option>
    <option label="August">August</option>
  </optgroup>
```

```

<optgroup label="Namen mit B">
  <option label="Berta">Berta</option>
  <option label="Barbara">Barbara</option>
  <option label="Bernhard">Bernhard</option>
</optgroup>
<optgroup label="Namen mit C">
  <option label="Caesar">Caesar</option>
  <option label="Christiane">Christiane</option>
  <option label="Christian">Christian</option>
</optgroup>
</select>
</p>
</form>

</body>
</html>

```



10.12: Netscape 6 und IE 6 zeigen die verschachtelten Auswahllisten als lange Liste mit Zwischenüberschriften an.

Für eine verschachtelte Menüstruktur definieren Sie Untergruppen von Auswahllisten. Dazu notieren Sie innerhalb von `<select>` und `</select>` für jede gewünschte Untergruppe ein `optgroup`-Element. Zwischen dem einleitenden `<optgroup>`-Tag und dem Abschluss-Tag `</optgroup>` notieren Sie jeweils eine Untergruppe von Einträgen. Im einleitenden `<optgroup>`-Tag vergeben Sie mit dem Attribut `label=` eine Menübeschriftung für die Gruppe von Einträgen. Dieser Eintrag wird beispielsweise als Menüeintrag mit Pfeiltaste rechts außen angezeigt, um dem Anwender zu signalisieren, dass sich hinter dem Menüeintrag ein Untermenü verbirgt.

Für die Einträge innerhalb einer Untergruppe notieren Sie jeweils ein `option`-Element. Auch hierbei können Sie den anzuzeigenden Menüeintrag mit dem Attribut `label=` bestimmen. Die Werte, die Sie hinter `<option>` notieren, sind für die Menüdarstellung ohne Belang. Es ist jedoch wichtig, sie zu notieren, denn Browser, die die Menüdarstellung nicht beherrschen, zeigen stattdessen eine gewöhnliche Auswahlliste mit den Einträgen an, die Sie hinter den `<option>`-Tags angeben.

**Beachten Sie:** Wenn Sie Unter-Untermenüs realisieren wollen, notieren Sie einfach innerhalb von `<optgroup>...</optgroup>` an der gewünschten Stelle einen weiteren Bereich mit `<optgroup>...</optgroup>`.

## 10.4 Radiobuttons und Checkboxes

### 10.4.1 Radiobuttons definieren

Radiobuttons sind eine Gruppe von beschrifteten Knöpfen, von denen der Anwender einen auswählen kann. Es kann immer nur einer der Radiobuttons ausgewählt sein. Der Wert des ausgewählten Radiobutton wird beim Absenden des Formulars mit übertragen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Hier wird abkassiert!</h1>

<form action="input_radio.htm">
<p>Geben Sie Ihre Zahlungsweise an:</p>
<p>
<input type="radio" name="Zahlungsmethode" value="Mastercard"> Mastercard<br>
<input type="radio" name="Zahlungsmethode" value="Visa"> Visa<br>
<input type="radio" name="Zahlungsmethode" value="AmericanExpress"> American Express
</p>
</form>

</body>
</html>
```



10.13: Bei Radiobuttons kann nur eine der gezeigten Optionen aktiv sein.

Radiobuttons werden durch `<input type="radio">` definiert (*input = Eingabe*). Jeder Radiobutton sollte einen internen Bezeichnernamen erhalten, und zwar mit dem Attribut `name=`. Alle Radiobuttons, die den gleichen Namen haben, gehören zu einer Gruppe, d.h. von diesen Buttons kann der Anwender genau einen markieren. Der vergebene Name wird bei CGI-Scripts benötigt, um auf die Daten des Eingabefeldes zugreifen zu können. Bei E-Mail-Formularen taucht der Name als Feldname auf. Und auch bei dem Formularfeldzugriff mit JavaScript ist der Name von Bedeutung. Für die Namensgebung gelten die in Kapitel 10.2.1 beschriebenen Konventionen.

Mit dem Attribut `value=` bestimmen Sie einen internen Bezeichnerwert für jeden Radiobutton (*value = Wert*). Wenn der Anwender das Formular abschickt, wird der Bezeichnerwert des markierten Button übertragen. Vor oder hinter dem `<input>`-Tag können Sie den Text notieren, der als Beschriftung der jeweiligen Option erscheint.

Wenn Sie eine der Auswahlmöglichkeiten vorselektieren wollen, dann notieren Sie in dem `<input>`-Tag des entsprechenden Radiobutton das allein stehende Attribut `checked`, also z.B.:

```
<input type="radio" name="Typ" value="Kassenpatient" checked>
```

Wenn Sie XHTML-konform arbeiten wollen, müssen Sie dieses Attribut in der Form `checked="checked"` notieren. Mehr als eine Auswahlmöglichkeit dürfen Sie bei Radiobuttons nicht vorselektieren.

## 10.4.2 Checkboxes definieren

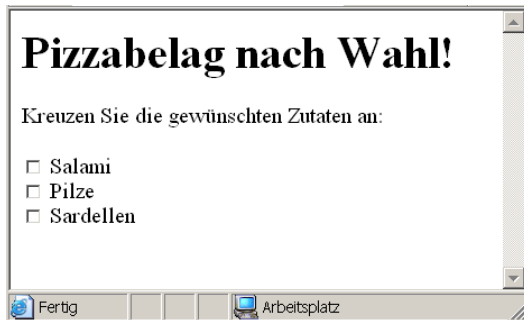
Checkboxes sind eine Gruppe von ankreuzbaren Quadraten, bei denen der Anwender keine, eins oder mehrere auswählen kann. Die Werte von ausgewählten Checkboxes werden beim Absenden des Formulars mit übertragen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Pizzabelag nach Wahl!</h1>

<form action="input_checkbox.htm">
<p>Kreuzen Sie die gewünschten Zutaten an:</p>
<p>
<input type="checkbox" name="zutat" value="salami"> Salami<br>
<input type="checkbox" name="zutat" value="pilze"> Pilze<br>
<input type="checkbox" name="zutat" value="sardellen"> Sardellen
</p>
</form>

</body>
</html>
```



10.14: Bei Checkboxes darf der Anwender beliebig viele der angebotenen Optionen ankreuzen.

Checkboxes werden durch `<input type="checkbox">` definiert (*input = Eingabe*). Jede Checkbox muss einen internen Bezeichnernamen erhalten, und zwar mit der Angabe `name=`. Alle Checkboxes, die den gleichen Namen haben, gehören zu einer Gruppe, d.h. von diesen Elementen kann der Anwender keines, eines oder mehrere ankreuzen. Für den Namen gelten die gleichen Bemerkungen wie bei Radiobuttons.

Mit dem Attribut `value=` bestimmen Sie einen internen Bezeichnerwert für jede Checkbox (*value = Wert*). Wenn der Anwender das Formular abschickt, werden die Bezeichnerwerte des oder der angekreuzten Buttons übertragen. Vor oder hinter dem `<input>`-Tag können Sie den Text notieren, der als Beschriftung der jeweiligen Option erscheint.

Wenn Sie Auswahlmöglichkeiten vorselektieren wollen, dann notieren Sie in dem `<input>`-Tag der entsprechenden Checkbox das allein stehende Attribut `checked`, also z.B.:

```
<input type="checkbox" name="Kenntnisse_in" value="HTML" checked>
```

Bei Checkboxes dürfen Sie mehrere Einträge vorselektieren. Wenn Sie XHTML-konform arbeiten wollen, müssen Sie dieses Attribut in der Form `checked="checked"` notieren.

## 10.5 Klick-Buttons

### 10.5.1 Klick-Buttons definieren (herkömmlich)

Sie können Schaltflächen definieren, die keine spezielle Bedeutung haben. Sinnvoll sind solche freien Klick-Buttons in Verbindung mit Script-Sprachen wie JavaScript. In der Regel werden sie dazu verwendet, Verweise oder andere JavaScript-gesteuerte Anweisungen auszuführen. Die hier beschriebene, herkömmliche Methode hat den Vorteil, dass sie auch von älteren Browsern (Netscape seit Version 2.x, MS Internet Explorer seit Version 3.x) interpretiert wird. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
```

```

<h1>Verweise einmal anders</h1>

<form action="input_button.htm">
<p>
<input type="button" name="Verweis"
      value="SELFHTML Portalseite"
      onClick="self.location.href='http://selfaktuell.teamone.de/'">
</p>
</form>

</body>
</html>

```



10.15: Ein klassischer Button, der beim Anklicken etwa eine Script startet.

Mit `<input type="button">` definieren Sie einen Button (*input = Eingabe*). Die Beschriftung des Button bestimmen Sie mit dem Attribut `value=` (*value = Wert*). Um anzugeben, was passieren soll, wenn der Button angeklickt wird, können Sie beispielsweise den Event-Handler `onClick=` verwenden. Als Wertzuweisung an das Event-Handler-Attribut können Sie dann JavaScript-Code notieren. Im obigen Beispiel wird mit Hilfe der JavaScript-Objekteigenschaft `location.href` ein Sprungverweis zu dem angegebenen URI ausgeführt. Eine andere beliebte, einfache JavaScript-Anweisung, mit der sich so ein Button belegen lässt, ist `onClick="history.back()"`. Damit erhält der Button die Zurück-Funktion im Browser.

Mit dem Attribut `name=` können Sie einen Namen für den Button vergeben. Unter diesem Namen ist der Button beispielsweise in JavaScript ansprechbar. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutschen Umlaute enthalten. Es gelten die in Kapitel 10.2.1 beschriebenen Regeln für die Namensgebung.

### 10.5.2 Klick-Buttons definieren (modern)

Ab HTML 4.0 dürfen anklickbare Buttons endlich so heißen wie sie heißen: nämlich **Button**. Solche Buttons sind flexibler als herkömmliche Buttons, denn sie dürfen auch einen definierten Inhalt haben, etwa eine Grafik. Der Nachteil ist, dass diese Sorte Buttons nur von neueren Browsern unterstützt wird – von Netscape 4.x noch nicht.

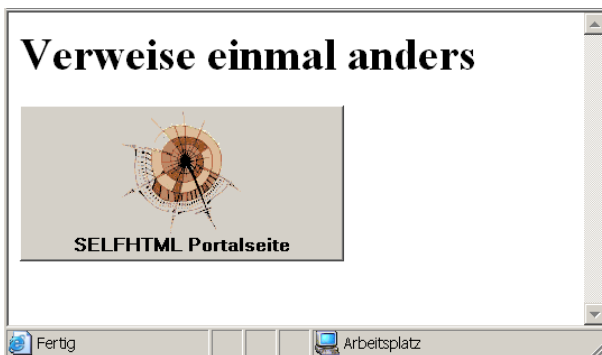
Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Verweise einmal anders</h1>

<form action="button.htm">
<div>
<button name="Klickmich" type="button" value="SELFHTML"
onClick="self.location.href='http://selfaktuell.teamone.de/'">
  <p><br>
  <b>SELFHTML Portalseite</b></p>
</button>
</div>
</form>

</body>
</html>
```



10.16: Mit dem `<button>`-Tag binden Sie auch Bilder in den Klick-Button mit ein.

Die Definition eines solchen Button leiten Sie mit `<button>` ein. Dieses Element hat ein End-Tag `</button>`, mit dem Sie die Definition des Button abschließen. Zwischen dem einleitenden Tag und dem End-Tag können Inhalte stehen. Alles, was Sie innerhalb von `<button>...</button>` notieren, wird als »Beschriftung« des Button angezeigt. Das dürfen durchaus auch eingebundene Grafiken sein, so wie im obigen Beispiel.

Im einleitenden `<button>`-Tag notieren Sie verschiedene Angaben zum Button. Etwas seltsam erscheint die Angabe `type="button"`, wo doch das Element schon so heißt. Notieren Sie diese

Angabe jedoch bei allen Buttons, die Sie für Script-Sprachen verwenden. Denn mit Hilfe des `button`-Elements können Sie auch zwei andere Button-Typen definieren, nämlich Buttons zum Absenden und solche zum Abbrechen (Kapitel 10.10).

Mit dem Attribut `name=` können Sie einen Namen für den Button vergeben. Für den Namen gelten die gleichen Anmerkungen wie im vorherigen Abschnitt *Klick-Buttons (herkömmlich)*.

Mit dem Attribut `value=` können Sie eine Beschriftung für den Button bestimmen, falls Sie keinen Inhalt innerhalb von `<button>...</button>` notieren. Beachten Sie jedoch, dass die Browser dies ignorieren und bei leerem Inhalt einen mickrigen leeren Button anzeigen. Um anzugeben, was passieren soll, wenn der Button angeklickt wird, können Sie beispielsweise den Event-Handler `onClick=` verwenden. Als Wertzuweisung an das Event-Handler-Attribut können Sie dann JavaScript-Code notieren.

**Beachten Sie:** Da der Button bereits anklickbar ist, sind alle HTML-Elemente darin verboten, die Verweis-Funktionen erzeugen. Es sind also keine Verweise beim Inhalt erlaubt, und Grafiken, die als Button-Inhalt angezeigt werden, dürfen kein Attribut `usemap=` für verweissensitive Flächen enthalten.

## 10.6 Felder für Datei-Upload

Diese Sorte Formularelement erlaubt dem Anwender, eine Datei von seinem lokalen Rechner zusammen mit dem Formular zu übertragen. Wenn ein CGI-Script die ankommenden Formulardaten auf dem Server-Rechner verarbeitet, ist es dadurch möglich, dem Anwender das Uploaden (Hochladen) von Dateien auf den Server-Rechner zu ermöglichen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Schicken Sie uns was Schickes!</h1>

<form action="input_file.htm" enctype="multipart/form-data">
<p>
W&auml;hlen Sie eine Textdatei (txt, html usw.) von Ihrem Rechner aus:<br>
<input name="Datei" type="file" size="50" maxlength="100000" accept="text/*">
</p>
</form>

</body>
</html>
```





10.17: Beim Datei-Upload wählt der Anwender mit dem typischen Auswahldialog seines Betriebssystems die Datei.

Mit `<input type="file">` definieren Sie ein Element für Datei-Upload (*input* = Eingabe, *file* = Datei). Der Web-Browser sollte dann ein Eingabefeld anzeigen, das die Eingabe einer Datei (in den meisten Fällen mit Pfadnamen) erlaubt. Rechts daneben sollte der Browser einen Button anzeigen, bei dessen Anklicken ein lokaler Dateiauswahl-Dialog am Bildschirm erscheint. Die Größe des Eingabefeldes (Anzahl Zeichen) können Sie mit `size=` bestimmen (*size* = Größe).

Wenn Sie das Attribut `maxlength=` angeben, sollte der Web-Browser die dahinter notierte Zahl als maximal erlaubte Dateigröße in Byte interpretieren (*maxlength* = maximal length = maximale Länge). Im obigen Beispiel wird auf diese Weise die Bytezahl auf 100000 Byte begrenzt. Wenn Sie `maxlength=` weglassen, kann der Anwender beliebig große Dateien senden. Beachten Sie jedoch, dass diese Angabe mit Vorsicht zu genießen ist. In der HTML-Version 3.2 wurde es so bestimmt, in der Version 4.0 wird diese Funktionalität beim `maxlength`-Attribut dagegen nicht mehr erwähnt. Verlassen Sie sich also nicht auf diese Angabe. Sicherer ist es, bei der Weiterverarbeitung mit einem CGI-Script im Script die Dateigröße zu ermitteln und das Script davon abhängig entscheiden zu lassen, ob die Datei akzeptiert oder verworfen wird.

Wenn Sie nur bestimmte Dateitypen zulassen wollen, können Sie mit der Angabe `accept=` die erlaubten Dateitypen eingrenzen (*accept* = akzeptieren). Hinter dem Gleichheitszeichen können Sie einen Mime-Type angeben (Anhang A.2). Dabei ist auch das Wildcard-Zeichen (\*) bei Subtypen erlaubt. Im obigen Beispiel werden mit `text/*` alle Textdateien akzeptiert. Dazu gehören reine Textdateien (\*.txt), aber z.B. auch HTML-Dateien (\*.html, \*.htm). Verlassen Sie sich aber auch bei dieser Angabe nicht darauf, dass der Browser vor dem Versenden des Formulars das tatsächlich prüft.

**Beachten Sie:** Wichtig ist, dass Sie im einleitenden `<form>`-Tag die Angabe `enctype="multipart/form-data"` notieren, wenn das Formular ein Element für Datei-Upload enthält.

Im Online-Angebot von SELFHTML (*selfaktuell.teamone.de/*) aktuell finden Sie einen Feature-Artikel, der beschreibt, wie das serverseitige Verarbeiten solcher Datei-Uploads aussehen kann: *Fileupload per Perl/CGI*. Um das dort beschriebene CGI-Script zu verstehen, benötigen Sie allerdings Kenntnisse in CGI/Perl.

Das Attribut `value=` in Verbindung mit `<input type="file">`, mit dem das Vorbelegen der Dateiauswahl möglich wäre, wird von moderneren Browsern aus Sicherheitsgründen nicht mehr unterstützt.

## 10.7 Versteckte Formularelemente

### 10.7.1 Versteckte Elemente definieren

Sie können Felder in einem Formular definieren, die dem Anwender nicht angezeigt werden. Versteckte Felder können Daten enthalten. Beim Absenden des Formulars werden die Daten versteckter Felder mit übertragen. Auf diese Weise können Sie beispielsweise zusätzliche Informationen an CGI-Scripts übergeben oder erläuternden Text einfügen, der bei der E-Mail-Übertragung der Formulardaten in der E-Mail mit enthalten ist.

Auch für JavaScript ist diese Möglichkeit interessant. Da ein JavaScript Formularfelder problemlos auslesen und deren Werte auch ändern kann, ist es auf diese Weise bequem möglich, interne Daten zu speichern, die nicht am Bildschirm angezeigt werden. So könnte ein JavaScript beispielsweise, nachdem die Seite mit dem Formular beim Anwender geladen ist, Informationen zur Bildschirmauflösung beim Anwender sammeln (siehe `Screen`-Objekt) und die Ergebnisse in ein verstecktes Formularfeld schreiben. Die Daten werden dann, wenn der Anwender das Formular abschickt, mit übertragen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Feedback</h1>

<form name="Feedback" action="http://www.der-allesrausbekommer.de/cgi-
bin/feedback.cgi">
<p>
<input type="hidden" name="UserBrowser" value="">
Ihr Name:<br>
<input name="UserName" type="text" size="40">
Ihr Kommentar:<br>
<textarea name="UserKommentar" rows="10" cols="40"></textarea>
```

```

</p>
</form>
<script type="text/javascript">
  document.Feedback.UserBrowser.value = navigator.userAgent;
</script>
</form>

</body>
</html>

```

Mit `<input type="hidden">` definieren Sie versteckte Felder in einem Formular (*input* = Eingabe, *hidden* = versteckt). Die Daten selbst bestimmen Sie beim Attribut `value` (*value* = Wert).

Im obigen Beispiel erhält das versteckte Formularfeld zunächst keine Daten (`value=""`). Unterhalb des Formulars ist jedoch ein JavaScript notiert. Dieses Script ermittelt den Browser-Typ, den der Anwender verwendet, und schreibt den ermittelten Wert in das versteckte Formularfeld. Wenn der Anwender das Formular absendet, wird also der verwendete Browser als Formularinhalt mit übertragen.

Auf die JavaScript-Anweisung wird hier nicht weiter eingegangen. In Kapitel 20 finden Sie die entsprechenden Befehle in der Objektreferenz beschrieben. Maßgeblich sind das `navigator`-Objekt und das `forms`-Objekt mit seinen Unterobjekten.

## 10.8 Gruppierung von Elementen und Label für Elemente

### 10.8.1 Elemente gruppieren

Größere Formulare bestehen häufig aus Gruppen von Elementen. Ein typisches Bestellformular besteht beispielsweise aus Elementgruppen wie »Absender«, »bestellte Produkte« und »Formular absenden/abbrechen«. Solche Elementgruppen können Sie ab HTML 4.0 eigens auszeichnen. Ein Web-Browser kann Elementgruppen durch Linien oder ähnliche Effekte optisch sichtbar machen. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>W&uuml;nschen Sie sich was!</h1>

<form action="fieldset_legend.htm">
<fieldset><legend><b>Absender</b></legend>
<table>
  <tr>

```

```

        <td>Ihr Vorname:</td>
        <td><input type="text" size="40" maxlength="40" name="Vorname"></td>
    </tr><tr>
        <td>Ihr Zuname:</td>
        <td><input type="text" size="40" maxlength="40" name="Zuname"></td>
    </tr>
</table>
</fieldset>
<fieldset>
<legend><b>Wunsch</b></legend>
<table>
    <tr>
        <td>Ihr Hauptwunsch:</td>
        <td><input type="text" size="40" maxlength="40" name="Hauptwunsch"></td>
    </tr><tr>
        <td>Ihr Zusatzwunsch:</td>
        <td><input type="text" size="40" maxlength="40" name="Zusatzwunsch"></td>
    </tr>
</table>
</fieldset>
</form>

</body>
</html>

```

10.18: Gruppierete Elemente umrahmt der Internet Explorer mit einer dünnen Linie.

Eine zusammengehörige Gruppe von Formularelementen schließen Sie in die Tags `<fieldset> ...</fieldset>` ein (*fieldset* = *Feldgruppe*). Dazwischen können Sie beliebige Teile Ihres Formulars definieren. Unterhalb des einleitenden `<fieldset>`-Tags und vor den ersten Formularinhalten der Gruppe sollten Sie ferner eine Gruppenüberschrift (Legende) für die Elementgruppe vergeben. Schließen Sie den Gruppenüberschriftentext in die Tags `<legend>... </legend>` ein. Den Text innerhalb davon können Sie mit Hilfe von HTML/CSS nach Wunsch formatieren.

## 10.8.2 Label für Elemente

Für Formularelemente wie Eingabefelder oder Auswahllisten gibt es normalerweise keine logische Beschriftungsmöglichkeit. Sie können zwar Text vor ein solches Element setzen wie »E-Mail-Adresse:«, aber ein solcher Text ist normaler HTML-Text, der keinen definierten Bezug zu dem Element hat, für das er als Beschriftung dient. Mit Hilfe von Labels können Sie jedoch einen solchen logischen Bezug zwischen Formularelement und Beschriftungstext herstellen. Ein Beispiel:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Meine Beschriftung geh&ouml;rt zu mir wie mein Name an der T&uuml;r!</h1>

<form action="label.htm">
<table>
  <tr>
    <td><label for="vorname">Vorname:</label></td>
    <td><input type="text" id="vorname" name="Vorname"></td>
  </tr><tr>
    <td><label for="nachname">Zuname:</label></td>
    <td><input type="text" id="nachname" name="Zuname"></td>
  </tr>
</table>
</form>

</body>
</html>

```

Das Beispiel zeigt, wie Sie mit Hilfe einer »blinden« Tabelle (Kapitel 8.6) Beschriftung und Eingabefelder eines Formulars ordentlich formatieren können. Das Besondere dabei ist, dass ein logischer Bezug zwischen Beschriftung und Eingabefeld hergestellt wird. Dazu muss das Formularelement selbst, im obigen Beispiel zwei Eingabefelder, mit Hilfe des Universalattributs `id=` einen eindeutigen Namen erhalten. Vergeben Sie keine zu langen Namen. Die Namen dürfen keine Leerzeichen und keine deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (`_`), den Bindestrich (`-`), den Doppelpunkt (`:`) oder den Punkt (`.`).

Den Beschriftungstext, den Sie einem solchen Formularelement zuordnen, notieren Sie als Inhalt zwischen `<label>` und `</label>`. Der Beschriftungstext kann auch jede Art von HTML/CSS-Formatierung enthalten. Das `label`-Element selbst hat keine sichtbare Wirkung am Bildschirm, es dient lediglich dem Zweck, den logischen Bezug zum Formularelement herzustellen. Diesen Bezug definieren Sie innerhalb des einleitenden `<label>`-Tags mit dem Attribut `for=` (*for = für*).

Dahinter geben Sie, in Anführungszeichen, exakt den id-Namen des Formularelements an, auf das sich das Label beziehen soll.

**Beachten Sie:** Das `label`-Element ist auch auf andere Formularelemente, zum Beispiel auf `textarea` oder `select`, anwendbar.

## 10.9 Tabulator-Reihenfolge, Tastaturkürzel und Ausgrauen

### 10.9.1 Tabulator-Reihenfolge

Modernere Browser erlauben es, mit Hilfe der -Taste an der Tastatur nacheinander die Elemente eines Formulars anzuspringen. Normalerweise werden die Formularelemente dabei in der Reihenfolge angesprungen, in der sie in der Datei definiert sind. Sie können jedoch eine andere Reihenfolge festlegen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Steppen Sie mit [Tab] durch!</h1>

<form action="tabindex.htm">
<pre>
Feld 1: <input type="text" size="40" maxlength="40" tabindex="4">
Feld 2: <input type="text" size="40" maxlength="40" tabindex="1">
Feld 3: <input type="text" size="40" maxlength="40" tabindex="3">
<input type="button" tabindex="2" value="Klick mich">
</pre>
</form>

</body>
</html>
```

Mit dem Attribut `tabindex=` in einem der Formular-Tags `<input>`, `<textarea>`, `<select>` oder `<button>` können Sie Angaben zur Tabulator-Reihenfolge machen. Notieren Sie die Angabe in allen entsprechenden Tags des Formulars und vergeben bei jeder Angabe eine Zahl. Beim Anspringen der Formularelemente mit der Tabulator-Taste wird zuerst das Formularelement mit der niedrigsten Tabindex-Nummer angesprungen, danach dasjenige mit der zweitniedrigsten usw. und als Letztes dasjenige mit der höchsten Tabindex-Nummer. Im obigen Beispiel wird also zuerst Feld 2 angesprungen, dann der Button am Ende, dann Feld 3 und zuletzt Feld 1. Es sind Zahlen zwischen 0 und 32767 erlaubt.

**Beachten Sie:** Die Tabindizes beziehen sich stets auf die gesamte angezeigte Datei. Dabei werden auch Verweise, Verweisbereiche in verweissensitiven Grafiken und Objekte mit einbezogen.

Wenn Sie solche Elemente neben dem Formular auch noch in Ihrer Datei haben, sollten Sie die Tabulator-Reihenfolge für alle Elemente gemeinsam festlegen. Siehe auch in Kapitel 6.7 über *Tabulator-Reihenfolge bei Verweisen*.

## 10.9.2 Tastaturkürzel

Sie können dem Anwender anbieten, mit einem Tastendruck zu einem bestimmten Formularelement zu springen. Das Element wird damit angesprungen – »ausgeführt«, also zum Beispiel angeklickt oder angekreuzt – wird es dadurch nicht. Ankreuzen von Radiobuttons und Check-boxen funktioniert meist mit der Leertaste, und Anklicken/Ausführen von Buttons mit der -Taste. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

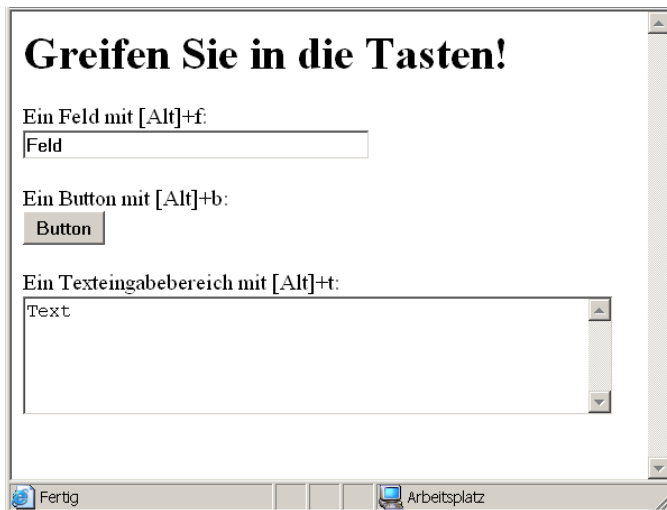
<h1>Greifen Sie in die Tasten!</h1>

<form action="accesskey.htm">
<p>Ein Feld mit [Alt]+f:<br>
<input type="text" size="40" value="Feld" accesskey="f"></p>
<p>Ein Button mit [Alt]+b:<br>
<input type="button" value="Button" accesskey="b"></p>
<p>Ein Texteingabebereich mit [Alt]+t:<br>
<textarea cols="50" rows="5" accesskey="t">Text</textarea></p>
</form>

</body>
</html>
```

Mit dem Attribut `accesskey=` können Sie ein Zeichen auf der Tastatur bestimmen, das der Anwender drücken kann, um ein Formularelement direkt anzuspringen (*accesskey* = *Zugriffstaste*). Bei den meisten Browsern ist es so geregelt, dass Sie bei `accesskey=` einen Buchstaben angeben können, der dann mit  und der Taste für den Buchstaben direkt anwählbar ist. Erlaubt ist das Attribut `accesskey=` in den Formular-Tags `<input>`, `<textarea>`, `<label>`, `<legend>` oder `<button>`.

**Beachten Sie:** Benutzen Sie Tasten, die auf jeder Tastatur zu finden sind. Funktionstasten etwa, wie es sie auf MF-II-Tastaturen für MS Windows gibt, können Sie mit dieser Angabe auch gar nicht ansprechen. Weisen Sie den Anwender im Text auf die Möglichkeit des Tastaturzugriffs hin.



10.19: Wenn Sie Formularfelder mit Hotkeys einsetzen, dann sollte die Beschriftung eindeutig darauf hinweisen.

### 10.9.3 Elemente ausgrauen

Sie können Elemente ausgrauen, um zu signalisieren, dass das Element im aktuellen Zusammenhang ohne Bedeutung ist. Ausgegraute Elemente werden auch bei Tabulator-Sprüngen übergangen. Sinnvoll ist diese Angabe allerdings vor allem als HTML-Grundlage für Script-Sprachen, die Elemente je nach Eingaben oder Auswahl in anderen Elementen dynamisch ausgrauen. So wäre es beispielsweise sinnvoll, Angaben zum Typ des gefahrenen Autos dynamisch auszugrauen, wenn der Anwender ankreuzt, dass er gar kein Auto besitzt. Als statische Angabe hat das Ausgrauen von Elementen dagegen wohl selten Sinn, denn Formularelemente, die man definiert, definiert man schließlich aus gutem Grund. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ein Formular f&uuml;r sp&auml;ter mal!</h1>

<form action="disabled.htm">
<p>Ihr Name:<br>
<input type="text" size="40" disabled></p>
<p>Ihr Kommentar:<br>
<textarea cols="50" rows="5" disabled></textarea></p>
</form>

</body>
</html>
```



10.20: Optisch ist nicht zu erkennen, dass das Eingabefeld deaktiviert (ausgegraut) ist.

Mit dem Attribut `disabled` grauen Sie ein Element aus. Erlaubt ist das Attribut in den Formular-Tags `<input>`, `<textarea>`, `<select>`, `<option>`, `<optgroup>` oder `<button>`. Beim Internet Explorer und bei Netscape ist der Effekt, dass das Element einfach nicht editierbar bzw. anklickbar ist. Eine optische Signalisierung gibt es dagegen in beiden Browsern nicht.

**Beachten Sie:** Wenn Sie XHTML-konform arbeiten wollen, müssen Sie dieses Attribut in der Form `disabled="disabled"` notieren. Weitere Hinweise im Kapitel 14.1.11 über *XHTML und HTML*.

## 10.10 Buttons zum Absenden oder Abbrechen

### 10.10.1 Buttons definieren (herkömmlich)

Zwei Standardbuttons stellt HTML zur Verfügung, um Formulareingaben zu handhaben: einen Button zum *Absenden* und einen zum *Abbrechen*. Mit dem Absende-Button kann der Anwender das ausgefüllte Formular losschicken. Mit den Formulardaten geschieht dann das, was im einleitenden `<form>`-Tag bei `action=` festgelegt wurde (siehe hierzu Abschnitt 10.1 *Formular definieren*). Mit dem Abbrechen-Button kann der Anwender alle Eingaben verwerfen. Das Formular wird nicht abgeschickt, Eingaben werden gelöscht. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Das hier k&ouml;nnen Sie absenden!</h1>

<p>Voraussetzung ist eine Verbindung ins Internet.
Es werden keine Daten gespeichert, das verarbeitende
CGI-Script gibt lediglich die eingelesenen Daten aus.</p>
```

```

<form action="http://selfhtml.teamone.de/cgi-bin/comments.pl">
<table border="0" cellpadding="5" cellspacing="0" bgcolor="#E0E0E0">
  <tr>
    <td align="right">Vorname:</td>
    <td><input name="Vorname" type="text" size="30" maxlength="30"></td>
  </tr><tr>
    <td align="right">Zuname:</td>
    <td><input name="Zuname" type="text" size="30" maxlength="40"></td>
  </tr><tr>
    <td align="right" valign="top">Kommentar:</td>
    <td><textarea name="Text" rows="10" cols="50"></textarea></td>
  </tr><tr>
    <td align="right">Formular:</td>
    <td>
      <input type="submit" value=" Absenden ">
      <input type="reset" value=" Abbrechen">
    </td>
  </tr>
</table>
</form>

</body>
</html>

```

**Das hier können Sie absenden!**

Voraussetzung ist eine Verbindung ins Internet. Es werden keine Daten gespeichert, das verarbeitende CGI-Script gibt lediglich die eingelesenen Daten aus.

Vorname:

Zuname:

Kommentar:

Formular: 

Fertig Arbeitsplatz

10.21: Ein vollständiges Formular mit den beiden Buttons zum Senden bzw. Abbrechen.

Mit `<input type="submit">` definieren Sie einen Absende-Button (*input = Eingabe, submit = bestätigen*). Beim Anklicken dieses Button werden die Formulardaten abgeschickt, und es wird die Adresse aufgerufen, die im einleitenden `<form>`-Tag beim Attribut `action=` angegeben ist.

Mit `<input type="reset">` definieren Sie einen Abbrechen-Button (*reset = zurücksetzen*). Eingegabene Daten werden verworfen und nicht abgeschickt. Mit dem Attribut `value=` bestimmen Sie die Beschriftung der Buttons (*value = Wert*).

**Beachten Sie:** Mit Hilfe von JavaScript können Sie die Eingaben von Anwendern vor dem Absenden des Formulars kontrollieren. In Kapitel 19.6 finden Sie das Anwendungsbeispiel *Formulareingaben überprüfen*.

### 10.10.2 Grafische Buttons definieren

Sie können innerhalb von Formularen Grafiken referenzieren und diese als Absende-Button definieren. Solche grafischen Buttons können Sie als Alternative zu Buttons vom Typ `<input type="submit">` verwenden. Ein Beispiel:

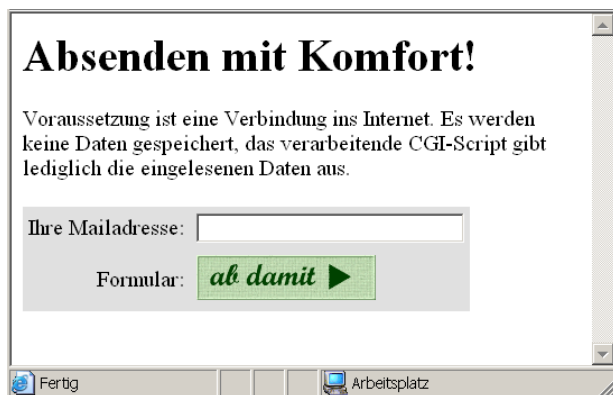
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Absenden mit Komfort!</h1>

<p>Voraussetzung ist eine Verbindung ins Internet.
Es werden keine Daten gespeichert, das verarbeitende
CGI-Script gibt lediglich die eingelesenen Daten aus.</p>

<form action="http://selfhtml.teamone.de/cgi-bin/comments.pl">
<table border="0" cellpadding="5" cellspacing="0" bgcolor="#E0E0E0">
  <tr>
    <td align="right">Ihre Mail-Adresse:</td>
    <td><input name="Mail" type="text" size="30" maxlength="30"></td>
  </tr><tr>
    <td align="right">Formular:</td>
    <td>
      <input type="image" src="absende.gif">
    </td>
  </tr>
</table>
</form>

</body>
</html>
```



10.22: Statt des typischen Button kann auch eine Grafik zum Absenden eines Formulars dienen.

Mit `<input type="image">` definieren Sie einen grafischen Button (*input = Eingabe*). Die gewünschte Grafikdatei bestimmen Sie mit dem Attribut `src=` (`src = source = Quelle`). Weisen Sie eine geeignete Grafik vom Typ GIF, JPEG oder PNG zu. Im obigen Beispiel wird vorausgesetzt, dass sich die Grafik `absende.gif` im gleichen Verzeichnis befindet wie die HTML-Datei mit dem Formular. Für das Adressieren der Grafik gelten die Regeln zum Referenzieren in HTML.

### 10.10.3 Buttons zum Absenden oder Abbrechen definieren (modern)

Dies funktioniert genauso wie in Kapitel 10.5.2 *Klick-Buttons definieren (modern)*. Dort wird erklärt, wie solche Buttons definiert werden. Um einen Button zum Absende-Button (Submit-Button) zu erklären, müssen Sie im einleitenden `<button>`-Tag `type="submit"` notieren. Um den Button zu einem Abbrechen-Button (Reset-Button) zu erklären, müssen Sie `type="reset"` notieren.

## 10.11 Formulare formatieren mit CSS

### 10.11.1 Formulare mit CSS gestalten

Mit CSS Stylesheets können Sie Formulare und Formularelemente optisch gestalten. Welche Gestaltungsmöglichkeiten funktionieren, ist jedoch noch ein Glücksspiel. Diejenigen Browser, die das Formatieren von Formularelementen mit CSS unterstützen, unterstützen es noch nicht konsequent. Ausprobieren ist also angesagt.

Für das gesamte Formular können Sie beispielsweise Angaben zur Schriftformatierung oder zum Hintergrund bestimmen. Für einzelne Formularelemente können Sie ebenfalls Angaben zur Schriftformatierung machen oder auch den vorgegebenen Feldrahmen abändern. Interessant sind zur Gestaltung von Formularen und Formularelementen folgende CSS-Eigenschaften: Schriftformatierung, Außenrand und Abstand, Innenabstand, Rahmen, Hintergrundfarben und -bilder.

Das folgende Beispiel zeigt ein vollständiges, absendbares Formular, das mit CSS Stylesheets gestaltet wurde:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Formulare und Formularelemente gestalten mit CSS</title>
<style type="text/css">
<!--
    form
        { background-image:url(background.gif); padding:20px; border:6px solid #DDDDDD;
    }
    td, input, select, textarea
        { font-size:13px; font-family:Verdana,sans-serif; font-weight:bold; }
    input, select, textarea
        { color:#0000CC; }
    .Bereich, .Feld
        { background-color:#FFFFAA; width:300px; border:6px solid #DDDDDD; }
    .Auswahl
        { background-color:#DDFFFF; width:300px; border:6px solid #DDDDDD; }
    .Check, .Radio
        { background-color:#DDFFFF; border:1px solid #DDDDDD; }
    .Button
        { background-color:#AAAAAA; color:#FFFFFF; width:200px; border:6px solid
#DDDDDD; }
-->
</style>
</head>
<body>

<h1>Das hier k&ouml;nnen Sie absenden!</h1>

<p>Voraussetzung ist eine Verbindung ins Internet.
Es werden keine Daten gespeichert, das verarbeitende
CGI-Script gibt lediglich die eingelesenen Daten aus.</p>

<form action="http://selfhtml.teamone.de/cgi-bin/comments.pl">
<table border="0" cellpadding="3" cellspacing="0">
<tr>
    <td align="right">Vorname:</td>
    <td><input name="Vorname" type="text" class="Feld" size="30" maxlength="30"></td>
</tr><tr>
    <td align="right">Zuname:</td>
    <td><input name="Zuname" type="text" class="Feld" size="30" maxlength="40"></td>
</tr><tr>
```

```

<td align="right" valign="top">Geschlecht:</td>
<td>
  <input type="radio" class="Radio" name="Geschlecht" value="m"> m&auml;nnlich<br>
  <input type="radio" class="Radio" name="Geschlecht" value="w"> weiblich
</td>
</tr><tr>
<td align="right" valign="top">Eigenschaften:</td>
<td>
  <input type="checkbox" class="Check" name="Eigenschaft" value="Raucher">
Raucher<br>
  <input type="checkbox" class="Check" name="Eigenschaft" value="Autofahrer">
Autofahrer<br>
  <input type="checkbox" class="Check" name="Eigenschaft" value="HTMLFreak"> HTML-
Freak
</td>
</tr><tr>
<td align="right" valign="top">N&auml;chste Gro&szlig;stadt:</td>
<td>
  <select name="Grossstadt" class="Auswahl" size="1">
    <option value="Hamburg">Hamburg</option>
    <option value="Berlin">Berlin</option>
    <option value="Frankfurt">Frankfurt</option>
    <option value="Muenchen">M&uuml;nchen</option>
  </select>
</td>
</tr><tr>
<td align="right" valign="top">Kommentar:</td>
<td><textarea name="Text" class="Bereich" rows="10" cols="50"></textarea></td>
</tr><tr>
<td align="right">Formular:</td>
<td>
  <input type="submit" class="Button" value="Absenden">
  <input type="reset" class="Button" value="Abbrechen">
</td>
</tr>
</table>
</form>

</body>
</html>

```

**Das hier können Sie absenden!**

Voraussetzung ist eine Verbindung ins Internet. Es werden keine Daten gespeichert, das verarbeitende CGI-Script gibt lediglich die eingelesenen Daten aus.

Vorname:

Zuname:

Geschlecht: ☐ männlich  
☐ weiblich

Eigenschaften: ☐ Raucher  
☐ Autofahrer  
☐ HTML-Freak

Nächste Großstadt:

Kommentar:

Formular:

10.23: Mit CSS Stylesheets gibt es kaum ein Detail im Formular, das Sie nicht ändern dürfen.

Im Beispiel werden die CSS-Formate für die Formularelemente zentral für eine HTML-Datei in einem Style-Bereich im Dateikopf definiert. Für die einzelnen Formularfeldtypen werden Formate für Klassen definiert – das sind im Beispiel die Einträge im Style-Bereich, die mit einem Punkt (.) beginnen.

Das Formular selber, also das `form`-Element, erhält im Beispiel eine Hintergrundgrafik zugewiesen, die ein Tapetenmuster bewirkt, sowie einen hellgrauen, 6 Pixel dicken Rahmen. Ein Innenabstand von 20 Pixeln sorgt für ein sauberes Aussehen. Für Tabellenzellen (`td`) und die Formularelemente `input`, `select` und `textarea` wird eine Schriftgröße von 13 Pixeln, die bevorzugte Schriftart Verdana und Fettschrift bestimmt. Für die drei Formularelementtypen wird ferner eine blaue Textfarbe festgelegt.

Eingabefelder und Eingabebereiche mit der Klasse `Eingabe` (die einleitenden HTML-Tags haben dazu das Universalattribut `class=` notiert) erhalten eine hellgelbe Hintergrundfarbe, eine feste Breite von 300 Pixeln, und der Rahmen um die Felder erhält die gleichen Eigenschaften wie der um das Formular. Auswahlelemente erhalten alle ein helles Zyan-Blau als Hintergrundfarbe, Auswahllisten der Klasse `Auswahl` ebenfalls eine feste Breite von 300 Pixeln. Bei den Auswahlelementen dieser Klasse wird versucht, einen hellgrauen Rahmen der Dicke 1 Pixel zuzuweisen.

Buttons schließlich, die im Beispiel für die beiden Buttons zum Absenden und Abbrechen benötigt werden, bekommen eine graue Hintergrundfarbe und weiße Schrift zugewiesen und sollen 200 Pixel breit sein. Auch die Buttons erhalten wieder den dicken hellgrauen Rahmen.

## 10.12 Formulare verarbeiten

### 10.12.1 Formulare als E-Mail

Viele Anwender, die als Zugabe zu ihrem Internet-Zugang vom Provider Platz von für eigene Web-Seiten erhalten, haben keine Möglichkeit, eigene CGI-Scripts zur Datenverarbeitung auf dem Server-Rechner einzusetzen. Dazu kommt, dass viele Homepage-Besitzer sich nicht in der Lage fühlen, sich in Programmiersprachen, fremde Betriebssystemumgebungen, Umgebungsvariablen und Datenübertragungsprotokolle einzuarbeiten. Dennoch möchten viele Seitenanbieter die Möglichkeit nutzen, mit Hilfe von HTML-Formularen gezielte und einheitlich strukturierte Daten von Besuchern ihrer Web-Seiten einzuholen. Zu diesem Zweck besteht die Möglichkeit, sich ausgefüllte Formulare als E-Mail zuschicken zu lassen.

Ein Problem stellen bei dieser Lösung jedoch Web-Browser dar, die nicht in der Lage sind, Formulare mit `action="mailto:..."` zu versenden. Dazu gehören Standalone-Browser, die kein eigenes E-Mail-Modul besitzen und darauf angewiesen sind, ein anderes E-Mail-Programm auf dem Rechner des Anwenders zu starten. Dabei klappt aber nicht unbedingt immer die Kommunikation zwischen Browser und E-Mail-Programm so, dass die Formulardaten übergeben und vom E-Mail-Programm anstandslos versendet werden. Bei vielen Anwendern tritt beispielsweise das Problem auf, dass das E-Mail-Programm einfach ein leeres Mail-Fenster öffnet, statt direkt die vom Browser übergebenen Formulardaten zu versenden. Gegen diese Probleme können Sie als Anbieter von Web-Seiten nichts unternehmen. Es ist also ein Glücksspiel, ob der Formularversand via E-Mail bei Ihren Seitenbesuchern klappt oder nicht. Um das Problem zu umgehen, können Sie einen öffentlichen CGI-Service für so genannte **Form-Mailer** in Anspruch nehmen. Dort werden die Formulardaten von einem CGI-Programm verarbeitet und Ihnen dann als E-Mail zugeschickt. Im Linkverzeichnis des Online-Angebots von SELFHTML ([selfaktuell.teamone.de/](http://selfaktuell.teamone.de/)) aktuell finden Sie auf der Seite Homepage-Services Verweise zu Anbietern solcher Form-Mailer.

Ein anderes Problem bei `mailto`-Formularen besteht darin, dass die Formulardaten beim Abschicken per Voreinstellung nach einem bestimmten Mime-Typ kodiert werden, dem Mime-Typ `application/x-www-form-urlencoded`. Dabei werden alle Leerzeichen, verschiedene Sonderzeichen und Umlaute durch spezielle Zeichenfolgen ersetzt. So lautet beispielsweise der Satz *Danke für die Hilfe! nach der Umwandlung: Danke+f%FCr+die+Hilfe%21*. Für Menschen ist das ziemlich ungenießbarer Lesestoff. Um die Kodierung zu verhindern, können Sie im einleitenden `<form>`-Tag zwar die Angabe `enctype="text/plain"` angeben. Von Anwendern, deren Web-Browser diese Angabe jedoch nicht interpretiert, werden Sie dennoch kodierte Formulardaten erhalten.

Benutzen Sie die Möglichkeit, sich Formulare direkt per E-Mail zuschicken zu lassen, also nur im Notfall und nur dann, wenn es nicht um wirklich wichtige Dinge geht. In allen anderen Fällen ist auf jeden Fall eine serverseitige Datenverarbeitung über ein CGI-Script oder vergleichbares



Script vorzuziehen. Denken Sie daran, dass die meisten Webspaces-Provider bereits vorgefertigte Form-Mailer-Scripts für ihre Kunden anbieten.

### 10.12.2 Formulare und CGI

Viele Anwendungsfälle für Formulare in HTML benötigen eine Datenverarbeitung auf dem Server-Rechner im Web. Wenn Sie beispielsweise Ihr gesamtes Web-Projekt nach einem Begriff durchsuchbar machen wollen, können Sie dem Anwender ein Formular anbieten, in dem er den Suchbegriff eingeben kann. Nach dem Abschicken des Formulars muss ein Script oder Programm auf dem Server-Rechner alle Dateien des Projekts nach dem gewünschten Begriff oder einen zuvor erzeugten Index oder eine Datenbank absuchen und eine dynamische HTML-Datei erzeugen, in der Verweise auf alle Seiten enthalten sind, die den Suchbegriff enthalten.

Für solche Fälle, aber auch für Anwendungsfälle wie automatische Gästebücher, Diskussionsforen, elektronische Einkaufskörbe usw. benötigen Sie ein Script oder Programm auf dem Server-Rechner, das die in einem Formular eingegebenen oder ausgewählten Daten sofort verarbeitet und dem Anwender echte Interaktion bietet.

Die Standardprogrammierschnittstelle auf Web-Servern hierfür ist CGI (Common Gateway Interface). Manche Web-Server haben – je nach installierter HTTP-Software – auch noch andere Programmierschnittstellen. Der Provider, bei dem Sie Ihre Web-Seiten speichern, muss Ihnen Zugriff auf diese Programmierschnittstelle erlauben, damit Sie solche Anwendungen selbst realisieren können. Viele Discount-Provider erlauben ihren Mitgliedern jedoch keinen solchen Zugriff.

Die meist verwendete Programmiersprache für CGI-Scripts ist Perl. Fragen Sie jedoch vor dem Programmieren eines eigenen CGI-Scripts Ihren Provider, welche Art von Programmen auf dem Server ausführbar sind. Wenn Sie in Perl programmieren wollen, fragen Sie den Provider, ob der Server über einen Perl-Interpreter verfügt und wo genau der Interpreter auf dem Server liegt.

# 11 Multimedia

## 11.1 Objekte einbinden

### 11.1.1 Allgemeines zu Objekten in HTML

Unter »Objekt« wird hier jede Art von Datei oder Datenquelle verstanden, die sich außerhalb einer HTML-Datei befindet und in diese HTML-Datei eingebunden werden soll. Es kann sich um eine Datendatei handeln, also etwa um eine Excel-Tabelle, eine AutoCad-Zeichnung, eine MIDI-Musikdatei, ein Flash-Movie, eine Streaming-Quelle für Rundfunkübertragung und vieles andere mehr. Es kann sich aber auch um eine vom Web-Browser ausführbare Datei handeln, also um ein Programm. Das können zum Beispiel Java-Applets oder ActiveX-Controls sein.

Damit nicht für jede neue Form der Einbindung anderer Ressourcen ein neues HTML-Element benötigt wird, soll ein einziges, mächtiges HTML-Element für alle Multimedia- und Fremdprogrammreferenzen genügen: das `object`-Element. Dieses Element kann zwar nicht das Problem lösen, wie eine beliebige Datei beim Anwender angezeigt werden kann, aber es bietet zumindest eine einheitliche Syntax und trägt dadurch zur Vereinfachung von HTML bei.

Das `object`-Element wird jedoch immer noch nicht vollständig von den Web-Browsern unterstützt. Das liegt zum Teil allerdings auch an der Vielzahl der proprietären Multimedia-Formate und ihren unterschiedlichen Schnittstellen. Das `object`-Element ist jedoch so ausgelegt, dass es möglichst flexibel auf alle denkbaren Anforderungen reagieren kann und dem Browser genügend Information liefert, um mit der ausführenden Anwendung zu kommunizieren und sie in sein Anzeigefenster einzubetten.

Objekte sind aus HTML-Sicht Inline-Elemente. In der »Strict«-Variante von HTML müssen solche Elemente innerhalb von Block-Elementen vorkommen, etwa in einem Textabsatz oder einem allgemeinen Bereich oder auch einer Tabellenzelle.

**Beachten Sie:** Wenn Sie mit Versuchen, das `object`-Element zu verwenden, keinen Erfolg haben, probieren Sie folgende herkömmlichen Möglichkeiten:

- Multimedia einbinden (Netscape, Kapitel 11.4)
- Java-Applets einbinden (Kapitel 11.3)

Das `object`-Element wird vom Internet Explorer seit Version 3.x und von Netscape seit Version 4.x unterstützt. Von einer ernst zu nehmenden Implementierung kann bei diesen Produktversionen aber noch keine Rede sein. Deshalb ist bei den Beispielen auf dieser Seite in der Regel Internet Explorer 5.0 und Netscape 6.0 angegeben. Aber auch das bedeutet noch lange nicht, dass mit diesen Browsern alles reibungslos funktioniert.

Binden Sie Multimedia-Dateien im Normalfall nicht kommentarlos in Ihre Web-Seiten ein, sondern weisen Sie den Anwender darauf hin, um welche Art von Daten es sich handelt und unter

welchen Voraussetzungen eine korrekte Anzeige möglich ist. Wenn Sie große Dateien einbetten, weisen Sie den Anwender im umgebenden Text auf die Größe der Datei hin.

### 11.1.2 Datendateien als Objekt einbinden

Sie können eine beliebige Datendatei als Objekt in eine HTML-Datei einbinden, also z.B. ein Video, eine Konstruktionszeichnung, eine als fertige Datei vorliegende 3-D-Welt, eine Musikdatei oder dergleichen. Auch einfache Textdateien und andere HTML-Dateien können Sie auf diese Weise einbinden. Ein Web-Browser kann solche Dateien anzeigen, wenn er entweder selber in der Lage ist, das Dateiformat anzuzeigen, oder wenn der Anwender ein entsprechendes Plug-In installiert hat. Wenn das Plug-In installiert ist, kann der Web-Browser die Datei so in seinem Anzeigefenster präsentieren, wie sie von dem Ursprungsprogramm erstellt wurde. Bei Abspielvorgängen, etwa von Videos oder Sound, wird ein entsprechender Player angezeigt – je nachdem, wie das Plug-In beschaffen ist.

Wenn dem Browser eine Verknüpfung zwischen dem Datentyp und einem Fremdprogramm bekannt ist, kann er das Fremdprogramm mit der betreffenden Datei starten. Ob die Daten dann jedoch innerhalb des Bereichs auf der Web-Seite angezeigt werden, der für das Objekt definiert ist, hängt davon ab, ob der Browser und das andere Programm entsprechende Kommunikationsmöglichkeiten, z.B. vom Betriebssystem bereitgestellte Kommunikationsschnittstellen, nutzen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Analoguhr als SVG-Grafik mit eingebettetem JavaScript</h1>

<p><object data="uhr.svg" type="image/svg+xml" width="200" height="200">
Ihr Browser kann das Objekt leider nicht anzeigen!
</object></p>

</body>
</html>
```

Mit `<object>` leiten Sie die Referenz auf die einzubindende Datei ein, mit `</object>` wird das Element beendet. Das `object`-Element hat keine Pflichtattribute. Je nachdem, was Sie damit einbinden, müssen Sie auf Herstellerhinweise achten oder selbst herausfinden, welche der nach HTML-Standard möglichen Attribute Sie angeben sollten oder müssen. Zwischen dem einleitenden `<object>`-Tag und dem abschließenden `</object>`-Tag kann »beliebiger« HTML-Quelltext notiert werden. Im obigen Beispiel ist einfach ein Text notiert, dass der Browser das Objekt nicht anzeigen kann. Ebenso gut könnte an dieser Stelle auch eine alternative Grafik notiert werden oder ein alternativer Versuch, das Objekt einzubinden, beispielsweise nach der herkömmlichen Netscape-Syntax (Kapitel 11.4). Es ist sogar erlaubt, als Inhalt eines `object`-

Elements ein anderes `object`-Element zu notieren, um damit beispielsweise ein alternatives Objekt in einem anderen Dateiformat einzubinden.



11.1: Dieses Beispiel wird nur von Internet Explorer ab Version 5 und Netscape 6 angezeigt.

Um eine Datendatei einzubinden, ist das Attribut `data=` vorgesehen. Damit referenzieren Sie die gewünschte Datei (*data = Daten*). Bei der Wertzuweisung an dieses Attribut gelten die Regeln zum Referenzieren in HTML. Wenn sich die Datei also beispielsweise im gleichen Verzeichnis wie die HTML-Datei befindet, dann genügt einfach die Angabe des Dateinamens – so wie im obigen Beispiel. Das Referenzieren mit relativen oder absoluten Pfadangaben ist jedoch ebenso möglich, z.B.:

```
<object data="verzeichnis/datei.xy">
<object data="../datei.xy">
<object data="../../woanders/datei.xy">
```

Auch das Referenzieren von einem vollständigen URI ist möglich, z.B.:

```
<object data="http://www.tolles-projekt.de/datei.xy">
```

Zusätzlich können Sie den Mime-Typ der Datei angeben. Bei Datendateien, die mit `data=` referenziert werden, notieren Sie dazu das Attribut `type=` – so wie im obigen Beispiel. Als Wert weisen Sie einen gültigen Mime-Type zu. Wenn Sie den Mime-Typ nicht kennen, lassen Sie die Angabe `type=` weg. Wenn Sie den Mime-Typ kennen und angeben, helfen Sie durch die Angabe dem Web-Browser, schneller zu reagieren.

Bei eingebundenen Objekten sollten Sie stets Angaben zu Breite und Höhe notieren. Mit dem Attribut `width=` legen Sie die Breite in Pixel oder in Prozent in Bezug auf die verfügbare Breite fest, und mit `height=` die Höhe.

- Manche Daten, zum Beispiel Videos, haben eine Originalbreite und Originalhöhe. Um so ein Video optimal einzubinden, sollten Sie dessen genaue Breite und Höhe kennen und mit `width=` und `height=` beim Einbinden der Videodatei angeben.
- Manche Plug-Ins, zum Beispiel Sound-Player, haben eine bestimmte Größe. Wenn Sie die genaue Größe kennen, sollten Sie beim Einbinden von Sound-Dateien, die mit dem betreffenden Plug-In wiedergegeben werden sollen, die genaue Höhe und Breite des Players angeben. So können Sie den Player vollständig und optimal innerhalb der Bildschirmanzeige der HTML-Datei platzieren. Die Größenangaben sind bei solchen Plug-Ins normalerweise bei der Software dokumentiert.

- Wenn Sie zunächst nicht wissen, wie viel Breite oder Höhe das Objekt benötigt, müssen Sie verschiedene Werte ausprobieren.
- Mit den Angaben `width="0"` und `height="0"` können Sie die sichtbare Anzeige auch ganz unterdrücken. Das kann zum Beispiel bei Hintergrundmusik erwünscht sein.

**Beachten Sie:** Das `object`-Element darf auch im Dateikopf einer HTML-Datei, also zwischen `<head>` und `</head>` notiert werden. Das ist dann sinnvoll, wenn die Datendatei nicht angezeigt werden soll, z.B. wenn einfach eine Hintergrundmusik abgespielt werden soll. In diesem Fall empfiehlt es sich, die Angaben `width="0"` und `height="0"` im einleitenden `<object>`-Tag zu notieren. Bei Objekten, die im HTML-Dateikopf eingebunden werden, darf jedoch zwischen `<object>` und `</object>` kein alternativer Inhalt stehen, der visuelle Ausgaben erzeugt.

Das W3-Konsortium sieht im HTML-Standard auch die Möglichkeit vor, das `object`-Element innerhalb eines Formulars zu notieren, um beispielsweise in visueller Interaktion mit dem Anwender Werte zu ermitteln, die dann beispielsweise in versteckten Formularfeldern gespeichert und zusammen mit den Formulardaten übertragen werden. Vorgaben, wie genau die Kommunikation zwischen Objekt und Formularelementen funktionieren soll, gehören jedoch nicht zur HTML-Spezifikation.

### 11.1.3 Verweissensitive Grafiken als Objekt einbinden

Diese Möglichkeit des `object`-Elements ist vor allem für Grafiken vorgesehen, die **nicht** einem der Standardtypen GIF, JPEG oder PNG entsprechen, die üblicherweise mit dem `<img>`-Tag eingebunden werden. Gedacht ist die Möglichkeit beispielsweise für vektorgrafische CAD-Zeichnungen oder andere Grafikformate, die Sie mit Hilfe des `object`-Elements einbinden. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<map name="Verweise">
<area shape="poly" coords="1,90,129,81,202,275,,273,415,202,417,150,311,1,173"
href="http://www.duesseldorf.de/">
</map>

<h1>Eine Stadt am Rhein</h1>

<p>Klicken Sie, falls Sie die Lurawave-Grafik sehen können, auf den Rhein.
Falls nichts passiert, unterstützt der Browser zwar die Anzeige des Objekts,
aber nicht das Feature verweissensitiver Grafiken für Objekte.</p>

<p><object data="ddorf.lwf" type="image/x-wavelet" width="413" height="417"
usemap="#Verweise">
```

```
Ihr Browser kann das Objekt leider nicht anzeigen!
</object></p>

</body>
</html>
```

Um eine verweissensitive Grafik als Objekt einzubinden, notieren Sie im einleitenden `<object>`-Tag das Attribut `usemap=`. Das restliche Verfahren ist dann das gleiche wie bei verweissensitiven Grafiken (siehe Kapitel 7.4). Das Beispiel benutzt für die Grafik das stark komprimierende Format Lurawave (Kapitel 24.1.5).

#### 11.1.4 Java-Applets als Objekt einbinden

Java-Applets sind ausführbare Programme, deren Bildschirmausgaben ein Web-Browser innerhalb seines Anzeigefensters darstellen kann. Applets können z.B. bewegte Animationen (Tricksequenzen) enthalten, Echtzeitabläufe in bewegten Grafiken darstellen (Stichwort: Börsenticker) oder Interaktionen mit dem Anwender austauschen. So werden Java-Applets etwa häufig beim Online-Banking eingesetzt.

Java-Applets müssen in kompilierter Form vorliegen, um bei der Referenzierung in einer HTML-Datei ausgeführt werden zu können. Normalerweise haben kompilierte Java-Applets die Dateinamenerweiterung `.class`. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Wagen Sie sich durchs Labyrinth!</h1>

<p><object classid="java:zmaze3d.class" codetype="application/java-vm" width="600"
height="400"></object></p>

</body>
</html>
```

Mit dem Attribut `classid=` im einleitenden `<object>`-Tag referenzieren Sie die Implementierung des JavaApplets (`classid = class identifier = Klassenbezeichner`). Erlaubt ist ein spezieller URI: Die Wertzuweisung besteht aus der festen Zeichenfolge `java:` – gefolgt von dem Namen der `.class`-Datei (also der ausführbaren Applet-Datei). Im obigen Beispiel wird auf diese Weise die Datei `zmaze3d.class` in der Form `classid="java:zmaze3d.class"` eingebunden.



11.2: Komplexe Interaktionen wie dieses Labyrinth sind mit Java-Applets kein Problem.

Wenn sich die Applet-Datei in einem anderen Verzeichnis befindet als die HTML-Datei, in der sie referenziert wird, oder auf einem anderen Internet-Server, dann müssen Sie den Pfad bzw. die Adresse des Server-Rechners und das Verzeichnis angeben, wo sich die Programmdatei befindet (nur das Verzeichnis, nicht mehr den Namen der Programmdatei). Dazu verwenden Sie – ebenfalls im einleitenden `<object>`-Tag – das Attribut `codebase=`. Bei der Wertzuweisung an dieses Attribut gelten die Regeln zum Referenzieren in HTML. Die Angabe von `codebase=` kann in einigen Fällen auch erforderlich sein, um Teile eines Java-Applets von einem bestimmten Internet-Server nachzuladen.

Ähnlich wie für Datendateien gibt es auch für ausführbare Programmdateien Mime-Typen. Bei Java-Applets sind die Angaben `codetype="application/java"` oder `codetype="application/java-vm"` gebräuchlich. Zwischen `<object>` und `</object>` können Sie im Zusammenhang mit Java-Applets Aufrufparameter notieren.

Viele Java-Applets erwarten einen oder mehrere solche Parameter beim Aufruf. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ein Lauftext mit Java</h1>

<p><object classid="java:zticker.class" codetype="application/java" width="600"
height="60">
<param name="msg" value="Die Energie des Verstehens">
<param name="href" value="../../index.htm">
```

```
<param name="speed" value="5">
<param name="bgco" value="255,255,255">
<param name="txtco" value="000,000,255">
<param name="hrefco" value="255,255,255">
</object></p>

</body>
</html>
```

Jeder Parameter wird durch `<param name= value=>` angegeben. Beim Attribut `name=` geben Sie den Namen des Parameters an. Bei `value=` geben Sie den gewünschten Wert des Parameters an, der dem Applet übergeben werden soll.

**Beachten Sie:** Bei Java-Applets, die aus mehreren *class*-Dateien bestehen, müssen Sie die Datei angeben, die den Programmstart enthält. Näheres dazu entnehmen Sie der Dokumentation, die solchen Java-Programmen beigelegt sein sollte. Auch die genauen Anweisungen zum Einbinden eines Java-Applets – z.B. zu erwarteten oder erlaubten Parametern – finden Sie normalerweise in der Dokumentation zu dem Java-Applet. Beachten Sie auch die herkömmliche Methode zum Einbinden von Java-Applets (Kapitel 11.3).

### 11.1.5 ActiveX-Controls als Objekt einbinden

ActiveX-Controls können ähnliche Aufgaben wahrnehmen wie Java-Applets. Jedoch sind sie stärker als Java-Applets in der Windows-Welt und der Welt von Microsoft verankert und werden nur vom Microsoft Internet Explorer interpretiert. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Eine MIDI-Datei</h1>

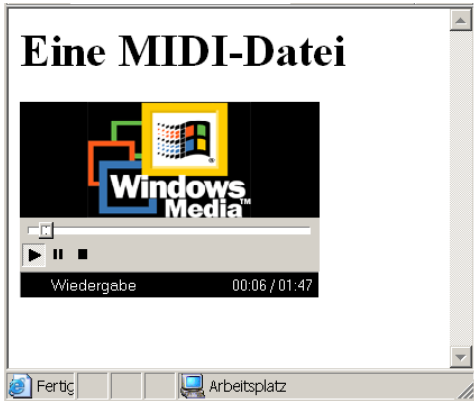
<object width="267" height="175"
    classid="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A">
  <param name="filename" value="ritmo.mid">
</object>

</body>
</html>
```

Mit dem Attribut `classid=` referenzieren Sie die Implementierung des gewünschten ActiveX-Controls (*classid = class identifier = Klassenbezeichner*). Die Angabe besteht aus der festen Zeichenfolge `CLSID:` – gefolgt von der Bezeichner-ID. Diese ID müssen Sie kennen. Im obigen Beispiel wird ein recht bekanntes ActiveX-Control referenziert, nämlich dasjenige, das unter



Windows zum Abspielen von Multimedia-Dateien zuständig ist. Es bindet den Media-Player von Windows in den Bereich des definierten Objekts ein. Mit `classid="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A"` binden Sie also ein ActiveX-Control ein, das es erlaubt, Sound- und Videodateien aller bekannten Formate wie WAV, AU, MID, MP3, AVI, MPEG usw. abzuspielen.



11.3: Per ActiveX-Control spielt der Internet Explorer ein Musikstück aus einer MIDI-Datei ab.

Mit dem Attribut `codebase=` können Sie die Internet-Adresse angeben, von der das ActiveX-Control geladen werden kann, falls der WWW-Browser es auf dem Rechner des Anwenders nicht findet (ActiveX-Controls werden nach dem Laden normalerweise in einem Windows-Unterverzeichnis abgespeichert – Dateien `.ocx` – und beim erneuten Aufruf auch wieder von dort geladen).

ActiveX-Controls können genauso wie Java-Applets Parameter übergeben bekommen. Diese werden auch genauso notiert, nämlich mit `<param name= value=>` zwischen `<object>` und `</object>`. Das ActiveX-Control im obigen Beispiel erwartet einen Parameter namens `file-name`, der bei `value=` die Angabe des gewünschten Dateinamens der abzuspielenden Datei erwartet.

### 11.1.6 Flash-Anwendungen als Objekt einbinden

Das Einbinden von Flash (Dateien `.swf`) funktioniert ähnlich wie das Einbinden von Java-Applets oder ActiveX-Controls. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Das gute alte Wurm-Spiel in Flash!</h1>
```

<p>Erst auf "Start" klicken. Dann Leertaste drücken. Mit dem Wurm die angezeigte Zahl ansteuern und treffen. Außenränder und Hindernisse nicht berühren!</p>

```
<object classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0"
width="600" height="400">
  <param name="movie" VALUE="nibbles.swf">
  <param name="quality" value="high">
  <param name="scale" value="exactfit">
  <param name="menu" value="true">
  <param name="bgcolor" value="#000040">
</object>

</body>
</html>
```



11.4: Auch mit Flash lassen sich komplexe Animationen wie etwa ganze Spiele realisieren.

Mit dem Attribut `classid=` referenzieren Sie die gewünschte Implementierung (*classid* = *class identifier* = *Klassenbezeichner*). Die Angabe besteht aus der festen Zeichenfolge `CLSID:` – gefolgt

von der Bezeichner-ID. Für Flash müssen Sie stets `classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"` notieren.

Mit dem Attribut `codebase=` können Sie die Internet-Adresse angeben, von der das Flash-Plug-In geladen werden kann, falls es beim Anwender nicht vorhanden ist. Im obigen Beispiel ist die Adresse für Flash 4 angegeben. Wenn Sie Flash-Anwendungen einsetzen, die mit Flash 5 erstellt wurden, geben Sie `codebase="http://active.macromedia.com/flash5/cabs/swflash.cab#version=5,0,0,0"` an.

Flash-Anwendungen können genauso wie Java-Applets Parameter übergeben bekommen. Diese werden auch genauso notiert, nämlich mit `<param name= value=>` zwischen `<object>` und `</object>`. Der wichtigste Parameter ist der namens `movie`, bei dem Sie bei `value` die gewünschte `swf`-Datei angeben.

In der Praxis ist es oft besser, zusätzlich zu dem `object`-Element auch noch die herkömmliche Netscape-Syntax mit anzugeben. So kommen auch ältere Netscape-Browser damit zurecht, sofern ein Flash-Plug-In installiert ist. Der Nachteil dabei ist jedoch, dass Sie ein HTML-Element verwenden müssen, das nicht zum HTML-Standard gehört. Sie erzeugen damit also ungültiges HTML. Für das obige Beispiel lautet dann der Quelltext:

```
<object classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0"
width="600" height="400">
  <param name="movie" VALUE="nibbles.swf">
  <param name="quality" value="high">
  <param name="scale" value="exactfit">
  <param name="menu" value="true">
  <param name="bgcolor" value="#000040">
  <embed src="nibbles.swf" quality="high" scale="exactfit" menu="false"
        bgcolor="#000000" width="600" height="400" swLiveConnect="false"
        type="application/x-shockwave-flash" pluginspage
="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash"></embed>
</object>
```

### 11.1.7 Rahmen um Objekte

Sie können festlegen, dass der Browser einen Rahmen um ein eingebundenes Objekt zeichnet. Das kann zur Verdeutlichung beitragen, dass es sich um ein eingebettetes Objekt handelt. Das Attribut dafür ist jedoch als *deprecated* gekennzeichnet und soll künftig aus dem HTML-Standard entfallen, da es durch die CSS-Eigenschaft `border` ersetzbar ist. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Rahmen um Objekte</title>
```

```

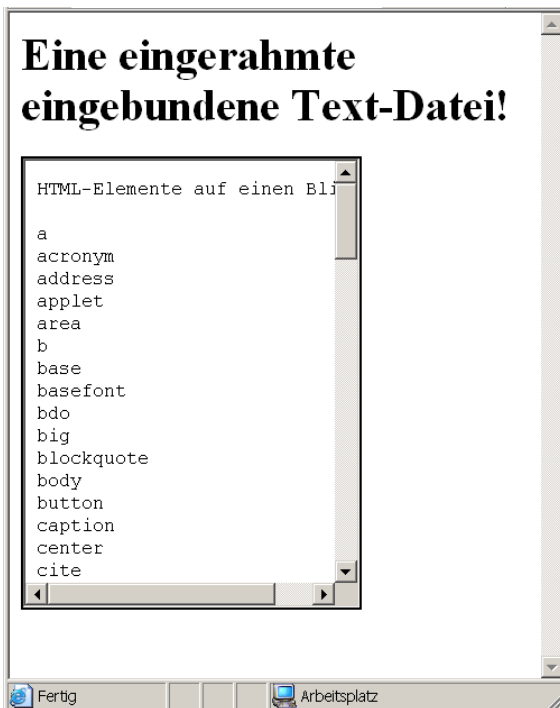
</head>
<body>

<h1>Eine eingerahmte eingebundene Text-Datei!</h1>

<p><object data="html.txt" type="text/plain" width="300" height="400" border="2">
Ihr Browser kann das Objekt leider nicht anzeigen!
</object></p>

</body>
</html>

```



11.5: Eine Textdatei als Objekt in einer Web-Seite (mit Rahmen).

Durch das Attribut `border=` innerhalb des einleitenden `<object>`-Tags wird ein Rahmen um das Objekt definiert (*border = Rand*). Die Angabe erfolgt in Pixeln und bedeutet die Rahmenstärke.

**Beachten Sie:** Das obige Beispiel zeigt, wie Sie einfache Textdateien »in a box« einbinden können – ebenso geht es mit HTML-Dateien und dem Mime-Typ `text/html`. Der Internet Explorer zeigt die Dateien an, allerdings keinen Rahmen. Netscape kann die Textdatei nicht als Objekt anzeigen.



- Mit `align="top"` wird der Text obenbündig zum eingebundenen Objekt ausgerichtet (*top = oben*).
- Mit `align="middle"` wird der Text mittig zum eingebundenen Objekt ausgerichtet (*middle = mittig*).
- Mit `align="bottom"` wird der Text untenbündig zum eingebundenen Objekt ausgerichtet (*bottom = unten*).

**Beachten Sie:** Um Objekte und Texte sauber zueinander auszurichten, können Sie auch mit rahmenlosen (»blinden«) Tabellen arbeiten.

Netscape interpretiert noch weitere Wertzuweisungen, die jedoch nicht zum HTML-Standard gehören und ungültiges HTML erzeugen:

- Mit `align="texttop"` richten Sie den Text obenbündig zum Objekt aus, und zwar an der Oberkante des kleinsten Textes (*texttop = oben am Text*).
- Mit `align="absmiddle"` richten Sie den Text absolut mittig zum Objekt aus, auch bei unterschiedlichen Textgrößen (*absmiddle = absolute middle = in jedem Fall mittig*).
- Mit `align="absbottom"` richten Sie den Text untenbündig zum Objekt aus, und zwar an der Unterkante des kleinsten Textes (*absbottom = absolute bottom = in jedem Fall untenbündig*).
- Mit `align="baseline"` richten Sie den Text genauso aus wie mit der Standardangabe `align="bottom"`. Benutzen Sie deshalb besser die Standardangabe.

### 11.2.2 Text um ein Objekt fließen lassen

Objekte können Sie links oder rechts ausrichten. Der umgebende Text fließt dabei um das Objekt. Mit zwei weiteren Attributen können Sie Abstand zum umfließenden Text erzeugen, damit der Text nicht direkt am Objekt »klebt«. Den automatischen Textumfluss können Sie auch abbrechen und die Textfortsetzung unterhalb des Objekts erzwingen. Alle dazu notwendigen Attribute sind allerdings als *deprecated* gekennzeichnet und sollen künftig aus dem HTML-Standard entfallen. Die gleiche Wirkung lässt sich nämlich auch mit CSS Stylesheets erzielen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#333333" text="#FFFFFF">

<h1>
<object classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0"
width="200" height="200" align="left" hspace="10" vspace="7">
<param name="movie" VALUE="neucler.swf">
```

```

</object>
Animation!
</h1>
<p>Kann schön sein, wenn sie nicht zu aufdringlich ist.
Dies ist ein Flash-Filmchen von gerade mal einem Kilobyte Größe.
<br clear="all"></p>
<h2>Gratulation!</h2>
<p>Der Text geht unterhalb weiter.</p>

</body>
</html>

```



11.6: Der Text fließt rechts um die Animation herum und wird dann unterhalb fortgesetzt.

Mit den Angaben `align="left"` bzw. `align="right"` veranlassen Sie den Browser, das Objekt linksbündig bzw. rechtsbündig auszurichten und den darauffolgenden Fließtext rechts bzw. links neben dem Objekt anzuzeigen und um das Objekt herumfließen zu lassen (*align = Ausrichtung, left = links, right = rechts*). Der umgebende Text kann verschiedene Elemente enthalten, zum Beispiel auch Überschriften oder Listen.

Um Abstand zwischen Objekt und umfließendem Text zu erzeugen, stehen die Attribute `hspace=` und `vspace=` zur Verfügung.

- Mit `hspace= [Pixel]` bestimmen Sie den Abstand zwischen Objekt und anderen Elementen links bzw. rechts davon (*hspace = horizontal space = horizontaler Abstand*). Mit `vspace= [Pixel]` bestimmen Sie den Abstand zwischen Objekt und anderen Elementen darüber bzw. darunter (*vspace = vertical space = vertikaler Abstand*).
- Das Attribut `hspace=` betrifft immer den linken und den rechten Randabstand vom Objekt, `vspace=` immer den oberen und den unteren Randabstand. Wenn Sie also beispielsweise bei einem links ausgerichteten Objekt `hspace=` definieren, wird nicht nur rechts vom Objekt zum nebenstehenden Text ein Abstand erzeugt, sondern auch links zum Rand des Anzeigefensters.

hin. Um dies zu verhindern, können Sie exakter arbeiten, indem Sie CSS Stylesheets verwenden.

- Sie können beide Attribute notieren oder auch nur eines davon.

Um den automatischen Textumfluss zu unterbrechen und zu erzwingen, dass alles was folgt unterhalb des Objekts angezeigt wird, können Sie einen Zeilenumbruch mit dem Attribut `clear=` notieren.

- Mit `<br clear="all">` erreichen Sie, dass der Textfluss ab der nächsten Zeile in jedem Fall unterhalb des Objekts fortgesetzt wird.
- Mit `<br clear="left">` erreichen Sie, dass der Textfluss ab der nächsten Zeile unterhalb eines linksbündig ausgerichteten Objekts fortgesetzt wird.
- Mit `<br clear="right">` erreichen Sie, dass der Textfluss ab der nächsten Zeile unterhalb eines rechtsbündig ausgerichteten Objekts fortgesetzt wird.

### 11.2.3 Meldungstext während des Ladevorgangs anzeigen

Multimedia-Objekte oder ausführbare Programme wie Java-Applets können zum Teil längere Ladezeiten verursachen. Auch dann, wenn die Dateien selbst nicht größer als durchschnittliche HTML- oder Grafikdateien sind, müssen beim Anwender in der Regel Plug-Ins oder Laufzeitinterpreter aktiviert werden. Das kostet Zeit.

Um den Anwender nicht im Unklaren über die Wartezeit und die Ladevorgänge an seinem Rechner zu lassen, können Sie einen Meldungstext angeben, der dem Anwender bis zur fertigen Anzeige des eingebundenen Objekts angezeigt wird. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Bauen Sie eine Brücke!</h1>
<p><object classid="java:Bridges.class" codetype="application/java-vm" width="350"
height="420"
standby="Das Spiel wird geladen. Bitte einen Moment Geduld...">
<param name="Color" value="white">
</object></p>
<p>Klicken Sie auf weiße Felder. Mit Rot versuchen Sie Ihre Brücke
zwischen
oberem und unterem Spielfeldrand zu bauen. Blau versucht Sie daran zu
hindern.</p>
</body>
</html>
```



Innerhalb eines einleitenden `<object>`-Tags können Sie mit dem Attribut `standby=` einen Meldungstext für den Ladevorgang angeben.

### 11.2.4 Objekte nur bei Aufruf durch Anwender anzeigen

Sie können ein Objekt definieren, das zunächst nur deklariert wird. Damit es angezeigt oder ausgeführt wird, muss es erst aktiviert werden. Das kann z.B. mit Hilfe eines Verweises auf das deklarierte Objekt geschehen oder durch ein anderes Objekt, das die Initialisierung anstößt. Browser, die diese Option nicht kennen (darunter auch Netscape, MS Internet Explorer), starten den Abspielvorgang sofort. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Wollen Sie oder wollen Sie nicht?</h1>

<p><object declare id="Marquee" classid="java:zticker.class"
codetype="application/java" width="600" height="60">
<param name="msg" value="Sie wissen was Sie wollen!">
<param name="href" value="../../index.htm">
<param name="speed" value="5">
<param name="bgco" value="255,255,0">
<param name="txtco" value="255,000,000">
<param name="hrefco" value="255,255,255">
</object></p>

<p><a href="#Marquee">Animierten Text anzeigen</a></p>

</body>
</html>
```

Innerhalb eines einleitenden `<object>`-Tags können Sie mit dem Standalone-Attribut `declare` ein Objekt als lediglich deklariert definieren (*declare = erklären, angeben*).

Mit dem Universalattribut `id=` können Sie einen Zielanker innerhalb des Objekts definieren, im Beispiel: »Marquee« (*id = Identifier = Bezeichner*). Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (`_`), den Bindestrich (`-`), den Doppelpunkt (`:`) oder den Punkt (`.`). Groß- und Kleinschreibung wird unterschieden.

Mit einem Verweis auf den Zielanker (siehe obiges Beispiel) können Sie den Anwender das Objekt aktivieren lassen. Beim `href`-Attribut notieren Sie das führende Gatterzeichen (`#`) und dahinter den `id`-Namen des Objekts.

## 11.2.5 Objekte formatieren mit CSS

CSS Stylesheets bieten Eigenschaften an, mit deren Hilfe die beiden Funktionen *Objekt zum umgebenden Text ausrichten* und *Text um Objekt fließen lassen* ebenso realisierbar sind wie mit HTML. Das »Fein-Tuning« ist mit Hilfe von CSS sogar deutlich genauer. Mit CSS müssen Sie arbeiten, wenn Sie die in diesem Abschnitt beschriebenen Effekte erzielen und dabei aber mit der HTML-Variante »Strict« arbeiten wollen. Maßgeblich sind im hier beschriebenen Zusammenhang folgende CSS-Eigenschaften: vertical-align (Vertikale Ausrichtung), float (Textumfluss), clear (Fortsetzung bei Textumfluss), Außenrand und Abstand. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Objekte formatieren mit CSS</title>
</head>
<body>

<h1>HIER SPIELT <object data="breeze.mid" type="audio/x-midi" width="200"
height="100"
style="vertical-align:middle; margin-left:30px; margin-right:30px;">
Ihr Browser kann das Objekt leider nicht anzeigen!
</object> DIE MUSIK</h1>

</body>
</html>
```

Mit `style="vertical-align:middle"` erreichen Sie in einem `<object>`-Tag das Gleiche wie mit `align="middle"`, mit `style="vertical-align:text-top"` das Gleiche wie mit `align="top"`, und mit `style="vertical-align:text-bottom"` das Gleiche wie mit `align="bottom"`.

Mit den CSS-Angaben `margin-left:30px;` und `margin-right:30px;` wird im obigen Beispiel rechts und links zum Text hin ein Abstand von 30 Pixeln geschaffen. Wenn Sie Text um ein Objekt fließen lassen wollen (im obigen Beispiel nicht realisiert), können Sie mit einer Angabe wie `style="float:left; margin-right:20px; margin-bottom:10px"` erreichen, dass das Objekt links ausgerichtet wird und nachfolgender Text rechts um das Objekt fließt. Dabei wird zwischen Objekt und Text ein Abstand von 20 Pixeln gehalten (links vom Objekt wird dagegen, anders als beim HTML-Attribut `hspace=`, kein Abstand von 20 Pixeln erzeugt). Nach unten hin, also zum Text unterhalb wird ein Abstand von 10 Pixeln definiert.

## 11.3 Java-Applets einbinden (herkömmlich)

### 11.3.1 Java-Applets einbinden

Java hat – zumindest im Web – nicht ganz das gebracht, was die Marketingstrategen anfangs prophezeit hatten. Aber eines kann man sagen: Es hat eine Menge Staub aufgewirbelt. Sun, der Entwickler von Java, hatte in den wilden Anfangsjahren von HTML zunächst eine eigene Syntax zum Einbinden von Java-Applets in seinen WWW-Browser »HotJava« integriert. Doch Netscape

implementierte eine andere Syntax. Diese Syntax wurde dann auch seinerzeit in den HTML-Standard 3.2 übernommen. Angesichts von Flash und all den anderen »Nachfolgern« von Java erkannte man aber, dass man eine allgemeinere »Media«-Syntax benötigte, und so entstand für HTML 4.0 das Konzept der Objekte mit dem `object`-Element, das unter anderem auch das Einbinden von Java-Applets erlaubt.

Vor allem aber wegen der immer noch nennenswerten Verbreitung von Netscape 4.x, dessen HTML 4.0-Unterstützung stark zu wünschen übrig lässt, ist jedoch bis heute noch die HTML 3.2-Syntax verbreitet und auch nur schwer tot zu kriegen. Bei vielen Dokumentationen zu Java-Applets wird diese Syntax wie selbstverständlich als die übliche genannt – und was die Rückwärtskompatibilität zu älteren Browsern betrifft, nicht ganz zu Unrecht. Dennoch sollte man als Web-Designer allmählich ein gewisses Bauchgrimmen bekommen bei der Verwendung dieser älteren Syntax. Denn im HTML-Standard ist sie auch längst als *deprecated* gekennzeichnet und soll künftig entfallen. Ein Beispiel:

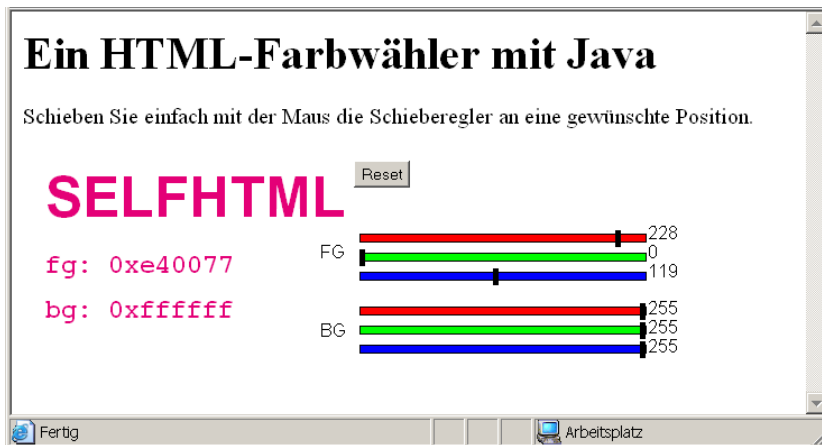
```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ein HTML-Farbwheel mit Java</h1>
<p>Schieben Sie einfach mit der Maus die Schieberegler an eine gewünschte
Position.</p>
<p>
<applet code="HexColor.class" width="640" height="200" alt="Java-Applets mit HTML-
Farbwheel">
<param name="TestText" value="SELFHTML">
</applet>
</p>

</body>
</html>
```

Mit `<applet>` leiten Sie die Referenz eines Java-Programms ein, mit `</applet>` wird sie abgeschlossen (*applet* = *application snippet* = *Anwendungsschnipsel*). Mit dem Attribut `code` referenzieren Sie das ausführbare Java-Applet, das normalerweise die Dateinamenerweiterung *.class* hat. Bei dieser Angabe dürfen Sie nur den Dateinamen notieren, auch wenn sich das Applet in einem anderen Verzeichnis oder auf einem anderen Server befindet.

Falls sich das Applet in einem anderen Verzeichnis als die einbindende HTML-Datei oder auf einem anderen Server befindet, können Sie mit `codebase` den URI des Verzeichnisses angeben. Mit `codebase=".."` beispielsweise adressieren Sie das Verzeichnis oberhalb des aktuellen Verzeichnisses usw. Wenn sich Ihr Java-Applet im gleichen Verzeichnis befindet wie die HTML-Datei, dann lassen Sie dieses Attribut weg.



11.7: Dieser Farbwähler zeigt die Hex-Notation passend zu den eingestellten RGB-Werten für Schrift und Hintergrund.

Für den Fall, dass das Java-Applet beim Anwender nicht ausführbar ist, können Sie mit der Angabe `alt=` einen Alternativtext definieren. Der Text muss in Anführungszeichen stehen. Er wird nur dann angezeigt, wenn das Applet nicht anzeigbar ist.

Während bei Grafiken die Angabe des `alt`-Attributs im HTML-Standard Pflicht ist und die Angabe von Breite und Höhe zwar empfohlen, aber nicht erzwungen wird, ist es beim `applet`-Element genau umgekehrt. Dort ist das `alt`-Attribut freiwillig, aber die Angaben zu Breite und Höhe sind Pflicht. Mit `width=` [Pixel/Prozent] bestimmen Sie die Anzeigebreite des Java-Applets, mit `height=` die Höhe (*width = Breite, height = Höhe*). Angegebene Prozentwerte beziehen sich auf die Breite bzw. Höhe des zur Verfügung stehenden Raums.

Zwischen dem einleitenden `<applet>`-Tag und dem abschließenden `</applet>` notieren Sie in vielen Fällen Parameter, die dem Applet beim Aufruf übergeben werden. Welche Parameter ein Java-Applet erwartet, entnehmen Sie der Dokumentation zu dem Applet. Jeder Parameter wird durch `<param name= value=>` angegeben (*param = parameter = Parameter, value = Wert*). Beim Attribut `name=` geben Sie den Namen des Parameters an. Bei `value=` geben Sie den gewünschten Wert des Parameters an, der dem Programm übergeben werden soll.

Mit dem Attribut `archive=` (im obigen Beispiel nicht verwendet) können Sie eine oder mehrere Archivdateien (z.B. `.zip`) als URI angeben. Verwenden Sie dieses Attribut nur, wenn es in der Dokumentation des Java-Applets verlangt wird.

Mit dem Standalone-Attribut `mayscript` (im obigen Beispiel nicht verwendet) erlauben Sie JavaScript, auf den Programmablauf des Java-Applets Einfluss zu nehmen. Mehr zu diesem Thema finden Sie im JavaScript-Kapitel im Abschnitt 20.10 *Zugriff auf Code in Java-Applets*. Dieses Attribut gehört jedoch nicht zum HTML-Standard.

Für den Zugriff mit JavaScript ist es auch sinnvoll, dem Java-Applet einen Namen zu geben. Dazu können Sie im einleitenden `<applet>`-Tag das Attribut `name=` notieren. Schließlich gibt es im HTML-Standard noch das Attribut `object=`, das ebenfalls im einleitenden `<applet>`-Tag notiert werden kann. Machen Sie davon jedoch nur Gebrauch, wenn es beim Java-Applet dokumentiert ist. Bei diesem Attribut kann ein Wert zugewiesen werden, der eine Ressource benennt,

mit deren Hilfe das Applet erst »zum Leben erwacht« (z.B. Freischalt-Funktion). Dies ist bei Applets der Fall, deren `init()`-Methode beim Start nicht ausgeführt wird und dazu eine externe Ressource benötigt.

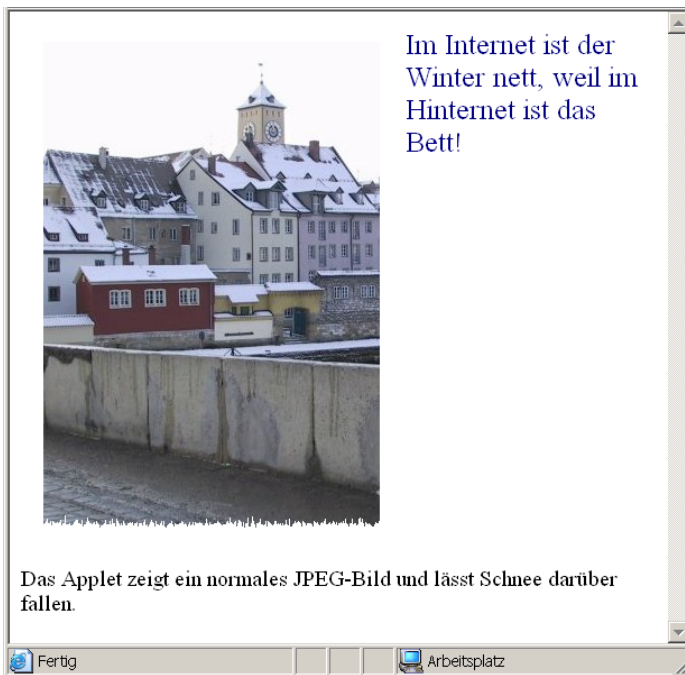
### 11.3.2 Java-Applets ausrichten

Java-Applets können Sie durch weitere Angaben im Verhältnis zum umgebenden Text ausrichten. Dabei können Sie den Abstand zum Text festlegen und auch die Textfortsetzung unterhalb des Applet-Bereichs erzwingen. Ferner können Sie auf den Programmablauf des Java-Applets Einfluss nehmen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<p style="color:#000080; font-size:26px">
<applet width="300" height="433" code="Nature.class" align="left" hspace="20"
vspace="12">
<param name="accumulate" value="yes">
<param name="background" value="FFFFFF">
<param name="backing" value="pixel.gif">
<param name="DeveloperInfo" value="Michael Chancey Jr.">
<param name="speed" value="75">
<param name="foreground" value="FFFFFF">
<param name="graphic" value="winterbild.jpg">
<param name="randomdir" value="yes">
<param name="season" value="winter">
<param name="fallcount" value="400">
</applet>
Im Internet ist der Winter nett, weil im Hinternet ist das Bett!<br clear="all">
</p>
<p>Das Applet zeigt ein normales JPEG-Bild und l&auml;u;ml;&szlig;t Schnee dar&uuml;ber
fallen.</p>

</body>
</html>
```



11.8: Mit diversen Attributen steuern Sie die Platzierung des Applets und den Textfluss.

Mit dem Attribut `align=` können Sie umgebenden Text vertikal zum Java-Applet ausrichten (*align = Ausrichtung*), und zwar mit `align="top"` obenbündig, mit `align="middle"` mittig und mit `align="bottom"` untenbündig. Ferner können Sie mit den Angaben `align="left"` bzw. `align="right"` den Browser veranlassen, das Java-Applet linksbündig bzw. rechtsbündig auszurichten und den darauffolgenden Fließtext rechts bzw. links neben dem Applet anzuzeigen und um das Objekt herumfließen zu lassen.

Um Abstand zwischen Applet und umfließendem Text zu erzeugen, stehen die Attribute `hspace=` und `vspace=` [Pixel oder Prozent] zur Verfügung. Um den automatischen Textumfluss zu unterbrechen und zu erzwingen, dass alles was folgt unterhalb des Applets angezeigt wird, können Sie einen Zeilenumbruch mit dem Attribut `clear="all"` (bei links- oder rechtsbündiger Ausrichtung), `clear="left"` (bei linksbündiger Ausrichtung) oder `clear="right"` (bei rechtsbündiger Ausrichtung) notieren.

## 11.4 Multimedia einbinden (Netscape herkömmlich)

### 11.4.1 Multimedia-Bereich definieren

In seiner damals Furore machenden Produktversion 2.0 führte Netscape neben anderen bahnbrechenden Neuerungen wie dem `font`-Element, den Frames und der Möglichkeit, Java-Applets einzubinden, auch ein Element namens `embed` ein, um andere Multimedia-Inhalte in HTML zu referenzieren. Damit einhergehend implementierte Netscape die so genannte Plug-In-Schnitt-

stelle in seinem Browser. Schnell entstand ein Wildwuchs an Plug-In-Software, mit deren Hilfe und dem hier beschriebenen Element sich beliebige multimediale oder proprietäre Binärformate in HTML einbinden ließen.

Aus heutiger Sicht gilt für dieses Element jedoch noch stärker als für das Element zum Einbinden von Java-Applets: Es ist ein Relikt und sollte allmählich fallen gelassen werden. Denn im Unterschied zum `applet`-Element hat das `embed`-Element nie zum HTML-Standard gehört. Es wird hier noch beschrieben, weil es heute leider immer noch sehr verbreitet ist und beispielsweise bei Netscape 4.x die einzig sinnvolle und halbwegs funktionierende Möglichkeit ist, Multimedia einzubinden. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Da kommt Freude auf!</h1>

<embed src="yippee.wav" width="140" height="60">

</body>
</html>
```

Mit `<embed src=>` definieren Sie eine Plug-In-Referenz (*embed* = *einbetten*, *src* = *source* = *Quelle*). Beim `src`-Attribut notieren Sie die gewünschte Datei. Dabei gelten die Regeln zum Referenzieren in HTML.

Mit den Attributen `width=` und `height=` sollten Sie stets die Größe des eingebundenen Bereichs bestimmen, damit der Browser die Seite schneller aufbauen kann. Beide Attribute erlauben als Wertzuweisung Pixel (absolut) oder Prozentangaben (relativ in Bezug auf die zur Verfügung stehende Fläche). Wenn Sie beide Werte auf 0 setzen, können Sie die sichtbare Anzeige des Bereichs unterdrücken – z.B. für Sounddateien, die automatisch abspielen. Das `embed`-Element kann mit oder ohne End-Tag notiert werden.

Ob und wie die Datei letztendlich beim Anwender »angezeigt« wird, hängt von vielen Faktoren ab, zum Beispiel von der Art der Datei, von den Hardware-Voraussetzungen des Anwenders, von den installierten Plug-Ins und von vorhandenen, dem Browser bekannten Fremdprogrammen. Bei Dateitypen, zu denen der Netscape-Browser ein Plug-In kennt, fragt er den Anwender, falls dieser das Plug-In nicht hat, ob er das Plug-In haben möchte, und erlaubt das Downloaden des erforderlichen Plug-Ins. Wenn der Web-Browser die Datei nicht verarbeiten kann, sollte er eine Fehlermeldung ausgeben und anstelle der Datei ein Symbol für nicht darstellbare Daten anzeigen.

Mit dem Attribut `pluginurl=` können Sie aber auch einen URI im Internet bestimmen, wo ein Plug-In zum Anzeigen oder Abspielen der eingebundenen Datei bereitsteht. Es muss sich nach dem Plug-In-Konzept von Netscape um eine Java-Archivdatei (Dateien *\*.jar*) handeln. Diese Angabe startet den in Netscape 4.x integrierten JAR-Installationsmanager (JIM). Der Anwender kann das Plug-In online von der angegebenen Adresse installieren. Netscape führt die Installa-

tion durch und kann anschließend sofort die eingebundene Multimedia-Datei anzeigen. Das Attribut `pluginurl=` ist erst seit Netscape 4.x verfügbar.

Mit `pluginspage=` können Sie einen URI bestimmen, wo Installationshinweise für ein Plug-In hinterlegt sind, das die eingebundene Multimedia-Datei anzeigen bzw. abspielen kann. Dies ist nicht ganz so elegant wie die Technik, die hinter `pluginurl=` steckt. Aber wenn das Plug-In in einem anderen Format als dem Java-Archivformat vorliegt, sollten Sie die Angabe `pluginspage=` wählen, sofern Ihnen eine Internet-Adresse mit einem geeigneten Plug-In bekannt ist.

Es gibt einen phantastischen Wildwuchs an weiteren Attributen für das `embed`-Element. Dies stammt alles aus einer Zeit, wo keinerlei Bewusstsein für die Notwendigkeit einer Standardisierung von HTML herrschte. Es liegt aber auch am schlecht durchdachten Konzept des `embed`-Elements. Um eingebundene Multimedia-Elemente zu steuern, bleibt eigentlich nur der Weg, dies über proprietäre Attribute zu tun. Eine kleine Auswahl:

- Mit `hidden="true"` können Sie die sichtbare Anzeige des eingebundenen Objekts verhindern. Das kann zum Beispiel sinnvoll sein, wenn Sie einen Sound abspielen wollen, aber keinen Sound-Player anzeigen wollen.
- Mit `autostart="true"` können Sie erzwingen, dass eine abspielbare Datei, also etwa ein Sound oder ein Video, sofort abgespielt werden, ohne dass der Anwender in der Abspiel-Software eine Funktion zum Start der Wiedergabe wählen muss.
- Mit `loop="true"` können Sie erzwingen, dass eine abspielbare Datei endlos wiederholt wird.
- Mit `palette="foreground"` können Sie erzwingen, dass das Plug-In bei der Anzeige die Vordergrundfarben des Anwenderrechners benutzt, mit `palette="background"` erzwingen Sie, dass das Plug-In die eingestellte Hintergrundfarbe benutzt.

All diese Attribute sind jedoch produktspezifisch und verschwinden genauso schnell wieder, wie sie entstanden sind.

#### 11.4.2 Multimedia-Bereiche ausrichten

Zum Ausrichten von Multimedia-Bereichen können Sie im `<embed>`-Tag die Attribute `align=`, `hspace=` und `vspace=` notieren. Die Wirkungsweise ist die gleiche wie im Abschnitt 11.2.1 *Objekt zum umgebenden Text ausrichten*.

#### 11.4.3 Alternativer Inhalt für Browser

Sie können einen Bereich definieren, der nur von Browsern angezeigt wird, die das `<embed>`-Tag nicht kennen. Netscape, der das `<embed>`-Tag interpretiert, ignoriert einen solchen Bereich. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
```



```
<h1>Einfach noch mal 'ne Runde &quot;nibbeln&quot; mit Flash!</h1>

<noembed>
<p>Sie verpassen was!</p>
</noembed>
<embed src="nibbles.swf" quality="high" scale="exactfit"
  menu="false" bgcolor="#000080" width="600"
  height="400" swLiveConnect="false"
  type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version
=ShockwaveFlash">

</body>
</html>
```

Mit `<noembed> . . . </noembed>` definieren Sie einen Bereich, der von Netscape ab Version 2.0 nicht angezeigt wird, wohl aber von Browsern, die das `<embed>`-Tag nicht kennen. Diese Browser kennen zwar auch das `<noembed>`-Tag nicht, aber sie stellen einfach alles dar, was dazwischen notiert ist. Es kann sich um Text, Grafik, Tabellen usw. handeln.

## 12 Layer (Netscape)

### 12.1 Layer definieren und positionieren

#### 12.1.1 Allgemeines zu Layern

Layer (*Schichten*) sind ein Sprachelement von HTML, das Netscape mit der Version 4.0 seines Browsers einführte. Mit Hilfe der beiden HTML-Elemente, die dazugehören, ist es möglich, beliebige Bereiche einer HTML-Datei als exakt positionierte Bereiche auszuzeichnen. So können Sie zum Beispiel eine Überschrift, eine Grafik und erklärenden Text dazu als Layer definieren. Diese definierte Einheit lassen Sie 100 Pixel von links und 50 Pixel von oben – gemessen am Anzeigefenster des Browsers – beginnen. Für die gesamte Einheit definieren Sie eine maximale Breite, zum Beispiel 200 Pixel.

In HTML allein haben die Layer jedoch nicht allzuviel Bedeutung, sieht man einmal von der Möglichkeit der absoluten Positionierung von Elementen ab. Der eigentliche Zweck der Layer ist es, die HTML-Grundlage für Dynamisches HTML nach dem Modell von Netscape zu bilden. Bei diesem Modell lassen sich Layer-Bereiche mit Hilfe von JavaScript auf vielfältige Weise verändern. So können Layer-Bereiche über den Bildschirm wandern, auf-/zuklappen, anderen Inhalt annehmen usw.

Sie können beliebig viele Layer innerhalb einer HTML-Datei definieren. Layer können sich dabei auch überlappen. Wenn Sie beispielsweise zwei Überschriften mit anschließender Grafik und erklärendem Text zu je einem Layer zusammenfassen, können beide Layer die gleichen Pixelpositionen erhalten. Mit Hilfe von JavaScript können Sie dann dynamisch einen der Layer ausschalten und den anderen einschalten. Sie können aber auch beide Layer gleichzeitig anzeigen und bestimmen, welcher über dem anderen liegen soll.

Layer können eigene Hintergrundfarben und sogar eigene Hintergrundbilder (Wallpapers) haben. Dadurch ergeben sich viele zusätzliche Gestaltungsmöglichkeiten beim Verteilen von Farben und Formen am Bildschirm. Wenn Sie eine GIF-Grafik mit transparentem Hintergrund als eigenen Layer definieren, können Sie diese Grafik über oder unter Texte oder andere Grafiken legen und dadurch interessante Effekte erzeugen.

Das einzige Problem mit den Layern ist: sie kamen zu spät. Das W3-Konsortium hatte sich bereits darauf eingeschossen, die Möglichkeit des absoluten Positionierens nicht in HTML, sondern in den CSS Stylesheets zu verankern. In den HTML-Sprachstandard 4.0 fanden die hier vorgestellten Elemente deshalb keinen Einlass. Angesichts der Tatsache, dass die neuen CSS-Eigenschaften zum Positionieren von Elementen offizieller Standard sind und sowohl von Netscape als auch vom MS Internet Explorer interpretiert werden, ist die Layer-Technik in HTML ins Abseits geraten. Wegen der starken Verknüpfung zwischen Layer-Technik und Dynamischem HTML nach Netscape 4.x werden sie an dieser Stelle jedoch noch beschrieben.

**Hinweis:** Diese HTML-Elemente gehören nicht zum HTML-Standard und funktionieren nur mit Netscape 4.x, sonst mit keinem Browser – auch nicht mehr mit Netscape 6.x.

### 12.1.2 Layer definieren

Sie können innerhalb des Körpers einer HTML-Datei fest positionierte Layer definieren. Solche Layer haben eine definierte obere linke Ecke im Anzeigefenster. Außerdem können Sie eine bestimmte Breite erzwingen. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<layer id="Beispiel_ID" name="Beispielname" top="150" left="250" width="300">

<h1>Irgendwo auf Ihrem Bildschirm ...</h1>
<p>Alles, was Sie hier sehen, ist in einen Layer
integriert. Auch das folgende Bild:</p>
<p></p>

</layer>

</body>
</html>
```



12.1: Ein Layer lässt sich irgendwo im Browser-Fenster absolut positionieren.

Mit `<layer>` leiten Sie die Definition eines Layer-Bereichs innerhalb einer HTML-Datei ein (*layer* = *Schicht*). Mit der Angabe `name=` können Sie einen Namen für den definierten Bereich vergeben. Zwischen dem einleitenden `<layer>`-Tag und dem abschließenden `</layer>` können Sie beliebige andere Elemente einer HTML-Datei notieren, also zum Beispiel Überschriften, Textabsätze, Tabellen, Grafik- oder Multimedia-Referenzen, Verweise usw. Alle Elemente, die Sie innerhalb eines Layers definieren, verhalten sich entsprechend den Eigenschaften, die Sie für den entsprechenden Layer definieren. Den Layer selbst müssen Sie sich dabei vorstellen wie ein eigenes Anzeigefenster ohne sichtbare Umrandungen, das in das normale Anzeigefenster des Web-Browsers eingebettet ist.

Mit dem Attribut `left=` bestimmen Sie die Anzahl Pixel vom linken Rand des Anzeigefensters, mit `top=` die Anzahl Pixel vom oberen Rand des Anzeigefensters (*left* = *links*, *top* = *oben*). Mit `width=` können Sie eine Anzeigebreite für den Layer-Bereich erzwingen, mit `height=` die Höhe (*width* = *Breite*, *height* = *Höhe*). Elemente innerhalb des Layers werden dann so umbrochen, dass die angegebene Breite eingehalten wird.

Mit `id=` können Sie einen Namen für den Layer-Bereich vergeben. Unter diesem Namen können Sie den Layer in JavaScript mit Hilfe des Objekts `layers` ansprechen. Neben `id=` wird bei Layern auch die Angabe `name=` für diesen Zweck interpretiert.

**Beachten Sie:** Wenn Sie mit Layern arbeiten und diese absolut im Anzeigebereich positionieren, wird es in den meisten Fällen sinnvoll sein, innerhalb einer HTML-Datei alle anzuzeigenden Elemente innerhalb von fest positionierten Layern zu notieren. Denn andernfalls gerät Ihnen die Anzeige leicht außer Kontrolle.

Wenn es der Inhalt erfordert, werden Angaben zur Breite außer Kraft gesetzt. Wenn beispielsweise eine Grafik oder eine Tabelle mehr Platz in Anspruch nimmt als die angegebene Breite, wird die Grafik bzw. die Tabelle in jedem Fall in voller Breite angezeigt. Dies können Sie allerdings verhindern – siehe *Anzeigebereich von Layern beschneiden* in Kapitel 12.1.4

Die Angaben zu linker und oberer Ecke sowie zur Breite sind zwar in den meisten Fällen erwünscht, jedoch nicht zwingend erforderlich. Wenn Sie keine Angaben dazu machen, beginnt der Layer einfach oben links, so wie das erste Element im Dateikörper der HTML-Datei.

### 12.1.3 Inline-Layer definieren

Inline-Layer beginnen relativ zu der Stelle, an der sie innerhalb der HTML-Datei stehen. Solche Layer erzeugen keinen eigenen Absatz im Fließtext. Ansonsten können sie aber alle Eigenschaften von Layern haben, also beispielsweise eine eigene Hintergrundfarbe. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

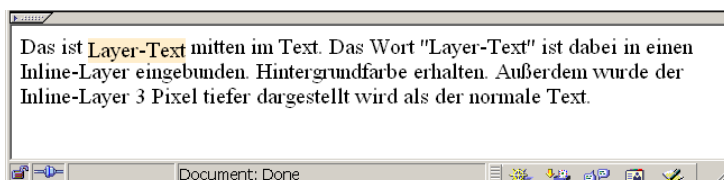
<p>Das ist <layer bgcolor="#FFEECC" top="3">Layer-Text</layer> mitten im
Text. Das Wort &quot;Layer-Text&quot; ist dabei in einen Inline-Layer
eingebunden. Hintergrundfarbe erhalten. Außerdem wurde der Inline-Layer 3
Pixel
```

```

tiefer dargestellt wird als der normale Text.</p>

</body>
</html>

```



12.2: Inline-Layer erscheinen wie Bilder innerhalb des Fließtextes an der Stelle, an der sie im Quelltext notiert sind.

Mit `<ilayer>` leiten Sie die Definition eines Inline-Layer-Bereichs ein, mit `</ilayer>` schließen Sie sie ab (*ilayer* = *inline layer*). Sie können auch Attribute wie `top=` oder `left=` auf einen `<ilayer>`-Bereich anwenden. Die Angaben werden dann jedoch nicht als Werte gemessen vom Fensterrand interpretiert, sondern als Werte gemessen von der normalen Position des Inline-Layers. So können Sie beispielsweise mitten in einem Text eine Anweisung notieren wie: `<ilayer top="3">Wort</ilayer>`. Dann erscheint dieses eine Wort um genau 3 Pixel tiefer als der übrige Text. Auch die Attribute `id=` und `name=` sind bei Inline-Layern erlaubt.

### 12.1.4 Anzeigebereich von Layern beschneiden

Sie können einen Bereich erzwingen, innerhalb dessen der Inhalt eines Layers angezeigt werden muss. Wenn der Inhalt mehr Platz in Anspruch nimmt (z.B. weil eine Grafik breiter ist), wird er abgeschnitten. Mit diesem Merkmal können Sie den Effekt erzeugen, dass der Inhalt eines Layers wie durch ein Fenster gesehen erscheint, wobei beim Blick durch das Fenster nur ein Ausschnitt des Inhalts sichtbar ist. Ein Beispiel:

```

<html>
<head>
<title>Text des Titels</title>
</head>
<body>

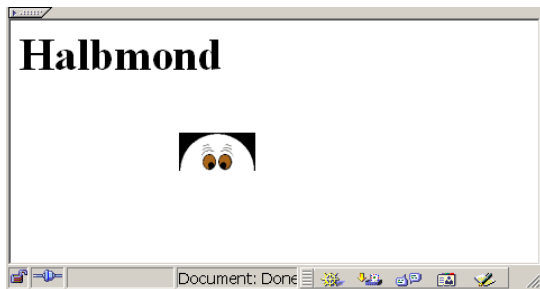
<h1>Halbmond</h1>

<layer top="100" left="150" clip="0,0,68,34">

</layer>

</body>
</html>

```



12.3: Mit dem Attribut `clip` beschneiden Sie den Anzeigebereich von Layers.

Mit dem Attribut `clip=` können Sie einen Anzeigebereich erzwingen (*clip* = Ausschnitt). Als Wert notieren Sie zwei oder vier Zahlenwerte zum Bestimmen des gewünschten Bereichs. Trennen Sie die Zahlenwerte durch Kommata.

Wenn Sie vier Zahlenwerte angeben, bedeuten die Zahlen die Koordinaten des gewünschten Rechtecks (in dieser Reihenfolge): `clip="Pixel für links, Pixel für oben, Pixel für rechts, Pixel für unten"`, und zwar relativ zur oberen linken Ecke des Layers. Wenn also im ersten der obigen Beispiele etwa durch `left="30"` und `top="20"` eine absolute obere linke Ecke für den Layer definiert wird, so bedeutet etwa die erste Angabe bei `clip=-20` – so viel wie: Der erzwungene Anzeigebereich beginnt 20 Pixel links vom linken Rand des Layers (der bei 30 Pixeln liegt), absolut gesehen also bei 50 Pixeln.

Wenn Sie zwei Zahlenwerte angeben, bedeuten die Zahlen die Breite und Höhe des gewünschten Rechtecks (in dieser Reihenfolge): `clip="Pixel für Breite, Pixel für Höhe"`, und zwar beginnend bei der oberen linken Ecke des Layers. Wenn Sie etwa wie im zweiten der obigen Beispiele mit `width=` eine bestimmte Breite des Layers erzwingen und sicherstellen wollen, dass die Breite auch eingehalten wird, wenn der Inhalt des Layers breiter ist, können Sie mit `clip="200,[Höhe]"` erzwingen, dass die Breite eingehalten wird.

### 12.1.5 Layer mit Inhalt aus anderer Datei

Es ist möglich, innerhalb eines Layers eine andere HTML-Datei anzuzeigen. Mit dem Attribut `src=` können Sie eine andere HTML-Datei in einen Layer einbinden. Dabei gelten die Regeln zum Referenzieren in HTML. Die Datei nimmt dabei den üblichen Raum im Anzeigefenster ein, beginnend bei den angegebenen Positionen zu `top=` und `left=`. Angaben zu `width=` und `height=` werden nur eingehalten, wenn der Inhalt der Datei in Breite und Höhe weniger Platz in Anspruch nimmt. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>SELFHTML im Layer</h1>
```

```
<layer top="80" left="40" src="../../index.htm"></layer>

</body>
</html>
```



12.4: Ein Layer kann den Inhalt einer HTML-Datei anzeigen.

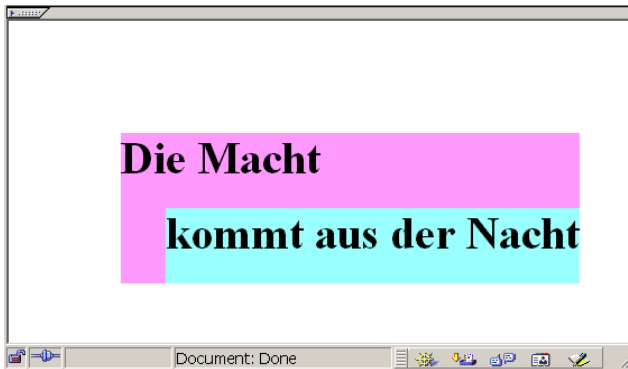
### 12.1.6 Layer verschachteln

Sie können innerhalb von Layer-Bereichen andere Layer definieren. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<layer top="100" left="100" bgcolor="#FF99FF">
<h1>Die Macht</h1>
  <layer pagex="80" left="40" bgcolor="#99FFFF">
    <h1>kommt aus der Nacht</h1>
  </layer>
</layer>

</body>
</html>
```



12.5: Ein Layer kann weitere Layer enthalten.

Wichtig ist dabei die unterschiedliche Wirkung der Positionsangaben. Im obigen Beispiel erhält der innere Layer die gleichen Angaben zu `left=` und `top=` wie der äußere Layer. Der äußere Layer ist dabei der Bezugspunkt für die Startposition des inneren Layers, nicht das Anzeigefenster! Der innere Layer beginnt im Beispiel also 100 Pixel von oben und 100 Pixel von links, gemessen an der linken oberen Ecke des äußeren Layers.

Es gibt jedoch auch die Möglichkeit, einen inneren Layer absolut zu positionieren. Dazu stehen die zusätzlichen Attribute `pagey=100` und `pagex=100` (*pagex = page x = Seite x-Wert, pagey = page y = Seite y-Wert*). Diese Angaben bewirken, dass als Bezugspunkt das Anzeigefenster und nicht der übergeordnete Layer verwendet wird.

### 12.1.7 Hintergrundfarbe für Layer

Sie können für jeden definierten fest positionierten oder fließenden Layer eine eigene Hintergrundfarbe definieren. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<layer top="0" left="0" width="50" height="40" bgcolor="#000000"></layer>
<layer top="0" left="50" width="50" height="40" bgcolor="#000033"></layer>
<layer top="0" left="100" width="50" height="40" bgcolor="#000066"></layer>
<layer top="0" left="150" width="50" height="40" bgcolor="#000099"></layer>
<layer top="0" left="200" width="50" height="40" bgcolor="#0000CC"></layer>
<layer top="0" left="250" width="50" height="40" bgcolor="#0000FF"></layer>

</body>
</html>
```



Mit dem Attribut `bgcolor=` im einleitenden Tag von `<layer>` oder `<ilayer>` bestimmen Sie die Hintergrundfarbe (*bgcolor = background color = Hintergrundfarbe*). Dabei gelten die Regeln zum Definieren von Farben in HTML.

**Beachten Sie:** Wenn Sie eine Hintergrundfarbe für einen Layer definieren, sollten Sie für Text, der in den Layern erscheint, eine kontrastierende Schriftfarbe wählen.

### 12.1.8 Hintergrundbild (Wallpaper) für Layer

Sie können für jeden definierten fest positionierten oder fließenden Layer ein eigenes Hintergrundmuster (Wallpaper) definieren. Das funktioniert genauso wie bei Hintergrundbildern (Wallpapers) für den gesamten Dateikörper. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body background="back1.jpg">

<layer top="0" left="0" width="400" height="300" background="back2.gif">
<h1>Bunte Zeiten</h1>
</layer>

</body>
</html>
```



12.6: Ein Layer kann auch ein individuelles Hintergrundbild haben.

Durch die Angabe `background=` im einleitenden Layer-Tag können Sie ein Hintergrundbild für den Layer bestimmen (*background = Hintergrund*). Dabei gelten die Regeln zum Referenzieren in HTML (Kapitel 3.7).

**Beachten Sie:** In diesem Buch finden Sie im Abschnitt 24.2 über typische Grafiksorten für Webseiten ein paar typische Hintergrundbilder. Netscape interpretiert auch animierte GIF-Grafiken als Hintergrundbilder.

## 12.2 Layer anordnen und kontrollieren

### 12.2.1 Angaben zur Schichtposition mit Layer-Namen

Wenn Sie mehrere Layer in einer HTML-Datei definieren, deren Anzeigebereiche sich überlappen, werden die Layer normalerweise in der Reihenfolge übereinander angezeigt, in der sie definiert werden. Sie können jedoch die Reihenfolge ändern und angeben, welcher andere Layer unmittelbar über oder unter dem aktuellen Layer liegen soll. Auf diese Weise können Sie das Überdecken von Layern kontrollieren. Voraussetzung ist, dass Sie für jeden Layer einen Namen vergeben. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<layer name="erster"
top="0" left="0" width="300" height="200" bgcolor="FFFF99">
<h1>ERSTER</h1>
</layer>
<layer name="zweiter"
top="50" left="100" width="300" height="200" bgcolor="99FF99">
<h1>ZWEITER</h1>
</layer>
<layer name="dritter" above="zweiter"
top="100" left="200" width="300" height="200" bgcolor="9999FF">
<h1>DRITTER</h1>
</layer>
<layer name="vierter" below="erster"
top="150" left="300" width="300" height="200" bgcolor="FF99FF">
<h1>VIERTER</h1>
</layer>

</body>
</html>
```



12.7: Die Reihenfolge der Layer in puncto Sichtbarkeit ist steuerbar.

Mit dem Attribut `above=` im einleitenden `<layer>`- oder `<ilayer>`-Tag können Sie bestimmen, dass ein zuvor definierter, benannter Layer unmittelbar über dem aktuellen Layer liegt (*above = oberhalb*). Das heißt, der aktuelle Layer soll bei Überschneidung von dem genannten Layer überdeckt werden.

Mit dem Attribut `below=` im einleitenden `<layer>`- oder `<ilayer>`-Tag können Sie bestimmen, dass ein zuvor definierter, benannter Layer unmittelbar unter dem aktuellen Layer liegt (*below = unterhalb*). Das heißt, der aktuelle Layer soll bei Überschneidung den genannten Layer überdecken. Die Layernamen hinter `above=` und `below=` sollten in Anführungszeichen stehen.

**Beachten Sie:** Bei `above=` und `below=` dürfen Sie nur Namen von Layern angeben, die in der Datei bereits zuvor definiert wurden. Die Angabe von Layern, die erst nach dem aktuellen Layer definiert werden, kann zu Anzeige Fehlern führen. Im ersten definierten Layer einer HTML-Datei sollten deshalb keine Angaben zu `above=` oder `below=` vorkommen.

Es ist nur je eine Angabe für `above=` und `below=` bei der Definition eines Layers erlaubt. Denn mit diesen Angaben legen Sie immer nur fest, welcher andere Layer direkt unter oder über dem aktuellen Layer.

### 12.2.2 Angaben zur Schichtposition mit Indexnummern

Wenn Sie mehrere Layer in einer HTML-Datei definieren, deren Anzeigebereiche sich überlappen, werden die Layer normalerweise in der Reihenfolge übereinander angezeigt, in der sie definiert sind. Sie können die Reihenfolge ändern, indem Sie für die einzelnen Layer Nummern vergeben. Layer mit höherer Nummer überdecken Layer mit niedrigerer Nummer. Ein Beispiel:

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
```

```

<layer z-index="3"
top="0" left="0" width="300" height="200" bgcolor="FFFF99">
<h1>ERSTER</h1>
</layer>
<layer z-index="1"
top="50" left="100" width="300" height="200" bgcolor="99FF99">
<h1>ZWEITER</h1>
</layer>
<layer z-index="4"
top="100" left="200" width="300" height="200" bgcolor="9999FF">
<h1>DRITTER</h1>
</layer>
<layer z-index="2"
top="150" left="300" width="300" height="200" bgcolor="FF99FF">
<h1>VIERTER</h1>
</layer>

</body>
</html>

```

Mit dem Attribut `z-index=` im einleitenden `<layer>`- oder `<ilayer>`-Tag bestimmen Sie die Reihenfolge der Layer bei Überlappungen (*z-index = Reihenfolge*). Geben Sie bei jedem definierten Layer `z-index=` und dahinter eine Nummer an. Der Layer mit der höchsten Nummer überdeckt alle anderen. Der Layer mit der niedrigsten Nummer wird von allen anderen überdeckt.

**Beachten Sie:** Bei den Angaben zu `z-index=` sind auch negative Werte erlaubt. Dadurch bewirken Sie bei Verschachtelungen, dass Layer, die innerhalb eines anderen Layers definiert werden, von ihren Eltern-Layern überdeckt werden. Bei positiven Zahlen ist dies umgekehrt.

### 12.2.3 Layer verstecken und anzeigen

Sie können bei der Definition von Layern festlegen, ob diese zunächst angezeigt werden sollen oder nicht. Das ist von Bedeutung, wenn Sie die Anzeige der Layer mit Hilfe von JavaScript dynamisch ein-/ausschalten wollen. Ein Beispiel:

```

<html>
<head>
<title>Text des Titels</title>
<script type="text/javascript">
<!--
function hideLayer(nr) {
  document.layers[nr].visibility = "hide";
}

function showLayer(nr) {
  document.layers[nr].visibility = "show";
}
//-->

```

```

</script>
</head>
<body>

<layer top="10" left="10" bgcolor="#DDFFDD">
<p><a href="javascript:showLayer(1)"><b>Anderen Layer anzeigen</b></a><br>
(dazu wird ein einfaches JavaScript eingesetzt)</p>
</layer>

<layer top="60" left="60" bgcolor="#CCEEFF" visibility="hide">
<p><a href="javascript:hideLayer(1)"><b>Diesen Layer verstecken</b></a><br>
(dazu wird ein einfaches JavaScript eingesetzt)</p>
</layer>

</body>
</html>

```



12.8: Die Sichtbarkeit von Layern lässt sich per JavaScript manipulieren.

Mit dem Attribut `visibility=` im einleitenden `<layer>`- oder `<ilayer>`-Tag können Sie die Anzeige des Layers explizit erzwingen oder verhindern (*visibility = Sichtbarkeit*).

- Mit `visibility="show"` legen Sie fest, dass der Layer in jedem Fall angezeigt wird (*show = zeigen*).
- Mit `visibility="hide"` legen Sie fest, dass der Layer nicht angezeigt, also versteckt wird (*hide = verstecken*).
- Mit `visibility="inherit"` legen Sie fest, dass der Layer nur dann angezeigt wird, wenn sein Eltern-Element ebenfalls angezeigt wird (*inherit = erben*). Diese Angabe ist nur bei inneren Layern in verschachtelten Layern von Bedeutung.

## 13 Universalattribute und weiterführende HTML-Elemente

### 13.1 Allgemeine Universalattribute

Unter »allgemeinen Universalattributen« werden hier solche Attribute verstanden, die in der Mehrzahl aller einleitenden HTML-Tags erlaubt sind und dort nicht als individuelle Attribute behandelt werden. Solche Attribute haben verschiedene Aufgaben. Es gibt beispielsweise Attribute, um einer bestimmten HTML-Auszeichnung einen individuellen Namen zu geben. Andere Universalattribute enthalten wichtige Angaben zur verwendeten Landessprache. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1 style="color:black" lang="de">Willkommen!</h1>
<h1 style="color:blue" lang="en">Welcome!</h1>
<h1 style="color:red" lang="fr">Bienvenue!</h1>
<h1 style="color:green" lang="it">Benvenuto!</h1>
<h1 style="color:olive" lang="pt">Boa vinda!</h1>

<p title="Der Text steht hier, im Titel!">Und der Text?</p>

</body>
</html>
```

Universalattribute notieren Sie wie gewöhnliche Attribute im einleitenden HTML-Tag. Im Beispiel werden mehrere Überschriften notiert. Alle haben ein Attribut `style=`, das CSS-Formatierungen für das entsprechende HTML-Element erlaubt. Ferner haben alle Überschriften ein Attribut `lang=`, das die Sprache des Inhalts bezeichnet. Der Textabsatz unten im Beispiel hat ein Attribut `title=`, das für Meta-Informationen zum Element gedacht ist. Neuere Browser zeigen solche Informationen in einem kleinen »Tooltip«-Fenster an, wenn der Anwender mit der Maus über den Inhalt des Elements fährt.



## Übersicht der Universalattribute

| Attribut | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Beispiel                                  |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| class=   | Gibt an, dass das HTML-Element einer bestimmten Stylesheet-Klasse angehört. Das Element übernimmt Formate, die zentral für diese Klasse definiert wurden. Einzelheiten dazu im CSS-Kapitel im Abschnitt 15.3.5 <i>Formate für Klassen definieren</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <code>&lt;h1 class="NeueSeite"&gt;</code> |
| dir=     | Gibt die Textrichtung der Landessprache an, die innerhalb des HTML-Elements verwendet wird. Interessant beispielsweise bei Sprachen wie Arabisch oder Hebräisch, wo die Textrichtung von rechts nach links geht. Erlaubt sind die Angaben <code>ltr</code> (links nach rechts) und <code>rtl</code> (rechts nach links).                                                                                                                                                                                                                                                                                                                                                                               | <code>&lt;td dir="rtl"&gt;</code>         |
| id=      | Ein dateiweit eindeutiger Bezeichnername für ein Element – wenn Sie Elemente damit auszeichnen, sollten Sie keinen <code>id</code> -Namen innerhalb einer HTML-Datei mehr als einmal vergeben. Interessant ist das <code>id</code> -Attribut beispielsweise im Zusammenhang mit neuem JavaScript nach DOM-Syntax, wo es die Objektmethode <code>getElementById()</code> gibt. Auch bei CSS Stylesheets spielt dieses Attribut eine Rolle, nämlich dann, wenn Sie zentrale Individualformate definieren (Kapitel 15.3.7). Achten Sie beim zugewiesenen Namen auf Groß-/Kleinschreibung! Für eine Sprache wie JavaScript sind Namen wie <code>Beispiel</code> und <code>beispiel</code> unterschiedlich. | <code>&lt;div id="Einleitung"&gt;</code>  |
| lang=    | Gibt die Landessprache an, die innerhalb des HTML-Elements verwendet wird. Interessant bei mehrsprachigen Dateien – aber auch für Suchmaschinen im Internet. Als Angabe ist ein standardisiertes Sprachenkürzel erlaubt, z.B. <code>de</code> für deutsch, <code>en</code> für englisch, <code>fr</code> für französisch, <code>it</code> für italienisch oder <code>es</code> für spanisch. Einzelheiten dazu und eine vollständige Übersicht finden Sie im Anhang A.3 <i>Sprachenkürzel</i> .                                                                                                                                                                                                        | <code>&lt;p lang="it"&gt;</code>          |
| style=   | Erlaubt es, den Inhalt eines Elements individuell mit CSS Stylesheets bzw. der im Dateikopf angegebenen Default-Sprache für Stylesheets zu formatieren. Einzelheiten dazu werden im CSS-Kapitel im Abschnitt 15.4 <i>HTML-Elemente direkt formatieren</i> behandelt.                                                                                                                                                                                                                                                                                                                                                                                                                                   | <code>&lt;p style="color:red"&gt;</code>  |
| title=   | Erlaubt es, HTML-Elemente mit kommentierendem Text bzw. Meta-Information auszustatten. Der kommentierende Text sollte vom Browser in einem kleinen Fenster (»Tooltip«-Fenster) angezeigt werden, wenn der Anwender mit der Maus über den Anzeigebereich des HTML-Elements fährt.                                                                                                                                                                                                                                                                                                                                                                                                                       | <code>&lt;p title="ironisch!"&gt;</code>  |

## 13.2 Event-Handler für Scripts

Event-Handler (*Ereignis-Behandler*) sind HTML-Attribute, die eine Verknüpfung zu einer Script-Sprache herstellen. Ein Ereignis ist zum Beispiel ein Mausklick oder ein Tastendruck des Anwenders. Ein Element, das einen Event-Handler enthält, kann auf ein solches Ereignis reagieren, wenn das Ereignis im Anzeigebereich des Elements stattfindet. Mit dem Event-Handler können Sie beispielsweise eine JavaScript-Funktion aufrufen, die das Ereignis verarbeitet und reagiert. Ein Beispiel:

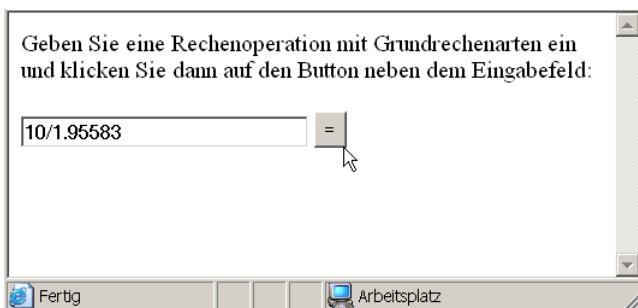
```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
<script type="text/javascript">
<!--
    function Ergebnis() {
        document.Formular.Eingabe.value = eval(document.Formular.Eingabe.value);
    }
//-->
</script>
</head>
<body>

<form name="Formular">
<p>Geben Sie eine Rechenoperation mit Grundrechenarten ein und klicken Sie dann
auf den Button neben dem Eingabefeld:</p>
<p>
<input type="text" name="Eingabe" size="30">
<input type="button" value=" = " onClick="Ergebnis()">
</p>
</form>

</body>
</html>

```



13.1: Über Event-Handler verknüpfen Sie Seitenelemente wie diesen Button mit einem Script.



Das Beispiel enthält ein Formular mit einem Eingabefeld und einem Klick-Button. Beim Klick-Button ist der Event-Handler `onClick=` notiert, der beim Anklicken des Buttons die JavaScript-Funktion `Ergebnis()` aufruft, die im Dateikopf in einem Script-Bereich notiert ist. Diese Funktion holt sich den Inhalt aus dem Eingabefeld, berechnet ihn und schreibt das Ergebnis wieder in das Eingabefeld.

**Beachten Sie:** Das obige Beispiel funktioniert auch mit älteren, JavaScript-fähigen Browsern. Solche Browser interpretieren Event-Handler nur in sehr eingeschränkter Form, nämlich bei Formularelementen und einigen wenigen anderen Elementen. Das Konzept der Event-Handler geht jedoch viel weiter. So kann beispielsweise auch eine Tabellenzelle beim Anklicken ihre Hintergrundfarbe wechseln usw. Solche Dinge funktionieren aber erst mit neueren Browsern (Internet Explorer ab Version 4.x, Netscape ab Version 6.x).



### Übersicht der universalen Event-Handler

Die folgende Tabelle listet nur solche Event-Handler auf, die im HTML-Standard als weitgehend universell und für die meisten HTML-Elemente einsetzbar festgehalten sind. Es gibt noch weitere Event-Handler, die jedoch nur für bestimmte Elemente gültig sind. Lesen Sie dazu auch den Abschnitt im JavaScript-Kapitel 18.10 über Event-Handler, wo Sie auch Beispiele für die einzelnen Handler finden.

| Attribut                  | Bedeutung                                                                                                 |
|---------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>onclick=</code>     | Für den Fall, dass der Anwender ein Element anklickt.                                                     |
| <code>ondblclick=</code>  | Für den Fall, dass der Anwender ein Element doppelt anklickt.                                             |
| <code>onmousedown=</code> | Tritt ein, wenn der Anwender die Maustaste gedrückt hält.                                                 |
| <code>onmouseup=</code>   | Tritt ein, wenn der Anwender die Maustaste gedrückt hat und sie nun wieder loslässt.                      |
| <code>onmouseover=</code> | Tritt ein, wenn der Anwender mit der Maus über ein Element fährt.                                         |
| <code>onmousemove=</code> | Tritt ein, wenn der Anwender die Maus bewegt, unabhängig davon, ob die Maustaste gedrückt ist oder nicht. |
| <code>onmouseout=</code>  | Tritt ein, wenn der Anwender mit der Maus über ein Element fährt und dieses dabei verlässt.               |
| <code>onkeypress=</code>  | Tritt ein, wenn der Anwender eine Taste drückt und diese gedrückt hält (siehe JavaScript-Kapitel)         |
| <code>onkeydown=</code>   | Tritt ein, wenn der Anwender, während er ein Element aktiviert hat, eine Taste drückt.                    |
| <code>onkeyup=</code>     | Tritt ein, wenn der Anwender eine Taste gedrückt hat und diese wieder loslässt.                           |

## 13.3 Weiterführende HTML-Elemente

### 13.3.1 Stylesheet-Bereiche definieren

Um mit Hilfe von Stylesheets zentrale Formate zu definieren, können Sie im Kopf einer HTML-Datei einen oder mehrere Stylesheet-Bereiche definieren.

Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
<style type="text/css" media="screen">
<!--
    body
    { background-image:url(back.jpg); padding:10px; }
    #GrosserText
    { font-size:150px; font-family:"Arial Black",Arial,sans-serif; color:#73FBE7; }
    #KleinererText
    { font-size:90px; font-family:"Arial Black",Arial,sans-serif; color:#FFFFFF; }
-->
</style>
</head>
<body>

<div id="GrosserText">CSS</div>
<div id="KleinererText">Stylesheets</div>

</body>
</html>
```



13.2: CSS Stylesheets bieten fast unbegrenzte Möglichkeiten.

Mit `<style>` leiten Sie innerhalb des HTML-Dateikopfs, also zwischen `<head>` und `</head>`, einen Bereich zum Definieren von Formaten ein (*style* = *Stil*), mit `</style>` beenden Sie den

Abschnitt. Die Formate, die Sie innerhalb des Bereichs nach der Syntax der Stylesheet-Sprache definieren, gelten dann für die gesamte Datei.

Innerhalb des einleitenden `<style>`-Tags müssen Sie mit dem Attribut `type=` angeben, welche Stylesheet-Sprache Sie innerhalb des Bereichs zum Definieren der Formate benutzen möchten. Die Wertzuweisung besteht im Mime-Type der gewünschten Stylesheet-Sprache. Die gängigste Angabe ist dabei `type="text/css"`. Mit dieser Angabe bestimmen Sie CSS Stylesheets als Sprache für die Formatdefinitionen. Sie können aber auch andere Sprachen benutzen. Falls Sie mehrere Style-Sprachen innerhalb der gleichen HTML-Datei einsetzen möchten, definieren Sie einfach für jede Sprache einen `style`-Bereich. Alle Bereiche müssen jedoch im Dateikopf stehen.

Mit dem Attribut `media=`, das im Gegensatz zu `type=` kein Pflichtattribut ist, können Sie den Typ des Ausgabemediums bestimmen. Neben der im obigen Beispiel angegebenen Wertzuweisung `screen` (für das Medium »Bildschirm«) gibt es weitere mögliche Angaben – siehe hierzu den Abschnitt 15.2.4 *Verschiedene separate Stylesheets für unterschiedliche Ausgabemedien*. Sie können, indem Sie mehrere `style`-Bereiche definieren, die jeweils unterschiedliche `media`-Attribute haben, Stylesheets für verschiedene Ausgabemedien definieren.

**Beachten Sie:** Es ist empfehlenswert, den Inhalt von Stylesheet-Bereichen in einen Kommentar (`<!-- bis -->`) zu setzen, so wie im obigen Beispiel. Dadurch verhindern Sie, dass ältere Browser die Inhalte irrtümlich als Text anzeigen. Auf die Stylesheet-Angaben wird hier nicht näher eingegangen. Lesen Sie dazu Kapitel 16.

## 13.4 Script-Bereiche in HTML

### 13.4.1 Script-Bereich definieren

Sie können im Dateikopf, also zwischen `<head>` und `</head>`, oder auch innerhalb des Dateikörpers, also zwischen `<body>` und `</body>`, einen oder mehrere Script-Bereiche definieren. Innerhalb von Script-Bereichen können Sie Anweisungen der verwendeten Script-Sprache notieren. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
<script type="text/javascript">
<!--
function Zeit() {
    var Jetzt = new Date();
    var Tag = Jetzt.getDate();
    var Monat = Jetzt.getMonth() + 1;
    var Jahr = Jetzt.getFullYear();
    if(Jahr < 999) Jahr += 1900;
    var Vortag = ((Tag < 10) ? "0" : "");
    var Vormon = ((Monat < 10) ? ".0" : ".");
    var Datum = Vortag + Tag + Vormon + Monat + "." + Jahr;
```

```
document.write("<h1>" + Datum + "<\\h1>")
}
//-->
</script>
</head>
<body>

<script type="text/javascript">
<!--
Zeit();
//-->
</script>
<p>Aktuelle Nachrichten ...</p>

</body>
</html>
```

Mit `<script>` leiten Sie einen Script-Bereich ein, mit `</script>` beenden Sie den Bereich. Innerhalb des Bereichs können Sie Anweisungen der verwendeten Script-Sprache notieren. Aus Sicht von HTML wird alles, was innerhalb des Bereichs steht, als »nackter Text« betrachtet. Dabei kann es allerdings zu Konflikten zwischen einer Sprache wie JavaScript und einem streng SGML-konformen HTML-Parser kommen. Ein solcher Parser betrachtet den Script-Bereich als beendet, sobald er auf die nächste Zeichenfolge `</` stößt. Wenn Sie innerhalb von JavaScript diese Zeichenfolge benötigen, wie im obigen Beispiel in der letzten Anweisung, wo mit Hilfe von `document.write` eine Überschrift erster Ordnung ins Dokument geschrieben wird. Am Ende des ausgegebenen Textes ist dort `<\\h1>` notiert statt `</h1>`. Der Backslash dient in JavaScript dazu, ein nachfolgendes Zeichen zu maskieren, was im Beispiel an der Stelle aus JavaScript-Sicht aber keine weitere Bedeutung hat und nichts weiter bewirkt. Der Backslash ist auch nicht für den JavaScript-Interpreter gedacht, sondern für penible HTML-Parser.

Um zu verhindern, dass ältere Browser den Inhalt des Script-Bereichs irrtümlich als Text ausgeben, empfiehlt es sich, den Inhalt in einen Kommentar zu setzen, so wie im obigen Beispiel. In der letzten Zeile des Script-Bereichs notieren Sie dann am besten einen einzeiligen JavaScript-Kommentar, eingeleitet durch `//`, gefolgt vom schließenden HTML-Kommentar `-->`. Der JavaScript-Kommentar ist erforderlich, um Fehlermeldungen in scriptfähigen Browsern zu unterdrücken. Berücksichtigen Sie, dass vor und nach dem HTML-Kommentar ein Zeilenumbruch zwingend erforderlich ist.

Innerhalb des einleitenden `<script>`-Tags geben Sie mit dem Pflichtattribut `type=` an, welche Script-Sprache Sie innerhalb des Bereichs benutzen möchten. Als Wert weisen Sie den Mime-Type der Script-Sprache zu. Die gängigste Angabe ist dabei `text/javascript` für JavaScript.

**Beachten Sie:** Das immer noch sehr viel genutzte Attribut `language=`, mit dem viele Web-Designer im einleitenden `<script>`-Tag die Script-Sprache angeben (z.B. `language="JavaScript"` oder `language="JavaScript1.2"`), gehört nicht zum HTML-Standard und erzeugt ungültiges HTML. Die Browser kennen das `language`-Attribut zwar und werten es aus, aber für modernere Browser reicht auch das Attribut `type=` zur Erkennung der Sprache aus. Auf die Script-Befehle wird hier nicht näher eingegangen. Lesen Sie dazu das JavaScript-Kapitel 17 und folgende.

### 13.4.2 Noscript-Bereich definieren

Sie können einen Bereich definieren, der nur angezeigt wird, wenn die verwendete Script-Sprache nicht verfügbar ist, das Script also nicht ausführbar ist. Dies ist der Fall, wenn der Browser die Script-Sprache nicht kennt oder wenn der Anwender das Interpretieren der Script-Sprache in den Einstellungen seines Browsers ausgeschaltet hat.

Wichtig ist eine solche Angabe beispielsweise, wenn Ihre Seiten intensiv JavaScript benutzen, um Inhalte anzuzeigen oder Verweise auszuführen. In solchen Fällen ist ein Projekt ohne JavaScript kaum nutzbar. Mit einem Noscript-Bereich können Sie einen entsprechenden Hinweis einbauen. Ein Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
<script type="text/javascript">
<!--
function Zeit() {
    var Jetzt = new Date();
    var Tag = Jetzt.getDate();
    var Monat = Jetzt.getMonth() + 1;
    var Jahr = Jetzt.getYear();
    if(Jahr < 999) Jahr += 1900;
    var Vortag = ((Tag < 10) ? "0" : "");
    var Vormon = ((Monat < 10) ? ".0" : ".");
    var Datum = Vortag + Tag + Vormon + Monat + "." + Jahr;
    document.write("<h1>" + Datum + "<\h1>")
}
//-->
</script>
</head>
<body>

<script type="text/javascript">
<!--
Zeit();
//-->
</script>
<noscript>
<h1>Guten Tag!</h1>
</noscript>

<p>Aktuelle Nachrichten ...</p>

</body>
</html>
```

Das Beispiel enthält genau die gleichen Scripts wie weiter oben bei *Script-Bereich definieren* beschrieben. Diesmal wurde jedoch zusätzlich an Browser gedacht, die kein JavaScript kennen oder bei denen JavaScript ausgeschaltet ist. Für solche Browser wird an der Stelle, an der bei script-fähigen Browsern die scripterzeugte h1-Überschrift mit dem aktuellen Datum eingesetzt wird, einfach eine gewöhnliche h1-Überschrift mit dem Inhalt `Guten Tag!` notiert.

Mit `<noscript>` leiten Sie einen Noscript-Bereich ein, mit `</noscript>` beenden Sie ihn. Dazwischen werden Block-Elemente erwartet, also beispielsweise Überschriften, Textabsätze oder `div`-Bereiche.

Web-Browser, die das `noscript`-Element kennen, zeigen den Inhalt dazwischen nur dann an, wenn der Anwender die benutzte Script-Sprache, etwa JavaScript, deaktiviert hat. Web-Browser, die gar keine Script-Sprachen kennen, kennen zwar auch das `noscript`-Element nicht, aber einer alten Regel gemäß ignorieren sie die Auszeichnung einfach und zeigen den Inhalt des Elements ganz normal an.

### 13.4.3 Weitere Angaben zum Script

Es gibt in HTML eine Reihe weiterer Optionen für Script-Bereiche, die als Attribute im einleitenden `<script>`-Tag notiert werden können:

- Mit dem Attribut `src=` können Sie ein Script einbinden, das in einer separaten Datei notiert ist. Das ist von Vorteil, wenn Sie ein Script auf vielen Seiten verwenden wollen. Dann brauchen Sie das Script nur noch zu referenzieren, statt es jedes Mal in der Datei zu notieren. Der Script-Bereich in der Datei kann in diesem Fall leer bleiben, aber auch noch zusätzlich ein lokales Script enthalten. Ein Beispiel für das Einbinden externer JavaScript-Dateien und genauere Hinweise dazu finden Sie im Abschnitt 17.3.3 *JavaScript in separaten Dateien*.
- Wenn Sie mit `src=` ein externes Script einbinden, können Sie zusätzlich mit dem Attribut `charset=` den Zeichensatz bestimmen, nach dem die externe Quelle behandelt werden soll. Das ist beispielsweise wichtig, wenn Sie in einem Script deutsche Umlaute und Sonderzeichen nicht maskieren, z.B. bei HTML-Ausgaben. In diesem Fall können Sie `charset="ISO-8859-1"` angeben, um dem Browser mitzuteilen, dass er im Zweifelsfall diesen Zeichensatz verwenden soll. Erlaubt sind Zeichensatzangaben, wie sie auf der Web-Adresse <http://www.iana.org/assignments/character-sets> angegeben sind.
- Ein weiteres Attribut namens `defer` (Standalone-Attribut) soll verhindern, dass ein Script dynamisch Text schreibt. Einzelne JavaScript-Anweisungen wie `document.write()` werden dadurch unterbunden. Beachten Sie, dass dieses Attribut bislang jedoch nur von neueren Versionen des MS Internet Explorers interpretiert wird.

## 14 XHTML und HTML

### 14.1 Unterschiede zwischen XHTML und HTML

#### 14.1.1 Allgemeines zu XHTML

Das Konzept der Auszeichnungssprachen, von denen HTML nur eine unter vielen, allerdings die bislang weltweit erfolgreichste und bekannteste ist, ist schon älter. Ein Merkmal des Konzepts solcher Auszeichnungssprachen ist, dass ihre Bestandteile nicht einfach »frei erfunden« sind, sondern dass es Meta-Sprachen gibt, mit deren Hilfe die Regeln und Bestandteile solcher Auszeichnungssprachen definiert werden.

Bereits seit 1986 gibt es die als ISO 8879 standardisierte Meta-Sprache **SGML** (*Standard Generalized Markup Language* = *standardisierte verallgemeinerte Auszeichnungssprache*). Diese Meta-Sprache erlaubt das Definieren von Auszeichnungssprachen mit Hilfe so genannter **DTDs** (*Document Type Definitions* = *Dokumenttyp-Definitionen*). In den DTDs wird festgelegt, welche Elemente eine Auszeichnungssprache hat, welche zugehörigen Attribute, sowie Regeln, welche Elemente innerhalb welcher anderen Elemente vorkommen können usw. Nun ist SGML sehr ausgereift, gilt aber gemeinhin als ziemlich »sophisticated«. So entschloss man sich dazu, eine reduzierte Variante von SGML zu etablieren, und zwar unter dem Namen **XML** (*Extensible Markup Language* = *erweiterbare Auszeichnungssprache*). XML erlaubt ebenso wie SGML das Definieren von Auszeichnungssprachen mit Hilfe von DTDs.

HTML, das Anfang der 90er Jahre entstand, wurde mit Hilfe von SGML definiert. Bis einschließlich HTML 4.x ist das auch heute noch der Fall. Im Zuge von XML und seiner wachsenden Bedeutung für immer mehr Dateiformate, die auch im Web ihren Einsatz finden, entstand der Wunsch, auch HTML mit Hilfe von XML zu definieren anstatt wie bisher mit SGML. Damit kein Versionenwirrwarr entsteht, entschloss man sich dazu, dieses neue, XML-basierte HTML mit einem neuen Namen und eigener Versionenkontrolle auszustatten. Heraus kam dabei **XHTML** (*Extensible HyperText Markup Language* = *erweiterbare Hypertext-Auszeichnungssprache*).

Seit Januar 2000 liegt XHTML 1.0 als Empfehlung des W3-Konsortiums vor und hat damit den gleichen verbindlichen Stellenwert wie etwa HTML 4.0. XHTML 1.0 ist nichts anderes als der Versuch, das SGML-basierte HTML 4.0 mit Hilfe von XML »nachzubauen«. Dahinter steckt keine Spielerei von Designern künstlicher Sprachen, sondern es gibt gute Gründe dafür:

- XHTML ist syntaktisch 100%ig kompatibel zu wichtigen XML-Standardsprachen wie SVG, WML, SMIL usw.
- XHTML kann dadurch als Dateninsel innerhalb anderer Sprachen eingebunden werden, und ebenso ist es möglich, diese anderen Sprachen wieder als Dateninsel innerhalb von XHTML einzubinden. So ist es beispielsweise problemlos möglich, eine SVG-Grafik als Dateninsel in

eine XHTML-Datei einzubetten. Die SVG-Grafik muss dadurch nicht mehr referenziert werden, sondern ist Bestandteil der XHTML-Datei.

- Durch die gemeinsame syntaktische Grundlage auf der Basis von XML ist es auch möglich, Script-Sprachen zu vereinheitlichen. Genau das geschieht beim **DOM** (*Document Object Model* = *Dokument-Objektmodell*). Das DOM definiert Objekte, Eigenschaften und Methoden für den Zugriff auf Elemente XML-basierter Sprachen. Möglich ist eine solche Anwendung aber nur, weil alle diese Sprachen denselben, XML-bedingten syntaktischen Grundmustern folgen. Da die syntaktischen Grundmuster von HTML sehr ähnlich sind wie die von XHTML, ist das DOM in der Praxis auch auf HTML anwendbar. Doch der Wunsch ist verständlich, ein HTML zu haben, das sich exakt an die Syntax von XML hält.
- Was für Script-Sprachen gilt, gilt ebenso auch für Stylesheet-Sprachen. Die selbst XML-basierte allgemeine Stylesheet-Sprache **XSL** (*Extensible Stylesheet Language* = *erweiterbare Formatsprache*) ist genau für diese Zwecke entworfen worden. Sie stellt eine einheitliche Grundlage für alle XML-basierten Sprachen zur Verfügung.

XHTML ist also XML-gerechtes HTML. Der »Nachbau« von HTML 4.0 in XHTML 1.0 ist so weit gelungen, dass es in XHTML die gleichen Elemente, Attribute und Verschachtelungsregeln gibt wie in HTML. Systembedingt durch die Syntax von XML gibt es jedoch diverse Unterschiede im Detail, die Sie kennen müssen, wenn Sie Ihre Web-Seiten in XHTML statt in HTML schreiben wollen.

Was die Browser-Kompatibilität betrifft, gibt es keine größeren Probleme. Browser, die mit HTML 4.0 zurechtkommen, verarbeiten auch Dokumente, die in XHTML geschrieben sind, und zeigen sie genauso am Bildschirm an wie HTML. Einzelne Probleme, besonders mit älteren Browsern, sind dagegen nicht auszuschließen.

### 14.1.2 Unterschied: Mime-Typen

Der Mime-Type für gewöhnliche HTML-Dokumente lautet stets `text/html`. XHTML-Dokumente können diesen Mime-Type ebenfalls haben. Gleichzeitig sind für XHTML-Dokumente jedoch auch die Mime-Typen `text/xml` oder `application/xml` erlaubt. Das W3-Konsortium ist darüber hinaus dabei, den Mime-Type `application/xhtml+xml` für XHTML zu etablieren. Für das Erstellen von XHTML-Dateien ist dies ohne Bedeutung. Wohl aber für die Kommunikation zwischen Web-Browser und Web-Server. Falls der Web-Server dem Web-Browser in einer HTTP-Umgebung die angeforderte XHTML-Datei als Datei des Typs `text/xml` oder `application/xml` anbietet, muss der Browser diesen Mime-Type akzeptieren und sich in der Lage fühlen, Dateien dieses Typs am Bildschirm anzuzeigen. Andernfalls kann es beispielsweise passieren, dass der Browser dem Anwender die Datei zum Download anbietet, statt sie ins Anzeigefenster zu laden.

### 14.1.3 Unterschied: Dateinamen

Neuere, XML-fähige Browser (Internet Explorer ab Version 5.x, Netscape ab Version 6.x) behandeln Dateien je nach Systemkonfiguration abhängig von der Dateinamenerweiterung unterschiedlich, wenn sie sonst keine Hinweise (Mime-Type) darauf haben, wie die Datei zu behandeln ist. Wenn Sie in einer Datei zwar alle Regeln von XHTML einhalten, die Datei aber



mit den typischen Endungen *.htm* oder *.html* abspeichern, benutzen diese Browser ihre HTML-Parser. Speichern Sie die Datei dagegen mit einer anderen Endung ab, z.B. *.xhtml*, dann benutzen die Browser unter Umständen ihre XML-Parser. Der Unterschied macht sich vor allem dann bemerkbar, wenn das Dokument syntaktische Fehler enthält. Die heute üblichen HTML-Parser der Browser lassen syntaktische Fehler im Dokument einfach durchgehen und versuchen das Dokument »irgendwie« anzuzeigen. XML-Parser brechen dagegen mit einer Fehlermeldung ab, sobald sie auf einen syntaktischen Fehler stoßen. Im Anzeigefenster des Browsers erscheint dann nur eine Fehlermeldung anstelle des Dokuments.

Bei der Darstellung XML-geparter, fehlerfreier XHTML-Dokumente ohne CSS oder andere Stylesheets reagieren die Browser unterschiedlich. Netscape 6.x zeigt das Dokument wie ein HTML-Dokument an, also mit den voreingestellten Darstellungen etwa für Überschriften, Textabsätze, Listen, Tabellen usw. Der Internet Explorer stellt solche Dokumente dagegen als reine Element-Baumstruktur dar. Erst wenn Sie die verwendeten Elemente mit CSS Stylesheets formatieren, zeigt der Internet Explorer das Dokument in nichtschematischer Form an.

Solange Sie also XHTML-konforme Dateien mit den HTML-typischen Endungen *.htm* oder *.html* abspeichern, erreichen Sie eine »normale« Darstellung im Browser, verhindern aber auch, dass die Dateien im Browser auf XML-Basis verarbeitet werden. Speichern Sie die Dateien dagegen unter einer anderen, dem Browser unbekannten oder mit dem Mime-Type `text/xml` verknüpften Dateiendung ab, dann werden die Dateien zwar auf XML-Basis verarbeitet, doch dann dürfen die Dateien tatsächlich keine syntaktischen Fehler enthalten, und wenn im Anzeigefenster des Internet Explorers etwas anderes als ein Strukturbaum erscheinen soll, müssen Sie Stylesheets verwenden, um die Elemente zu formatieren.

#### 14.1.4 Unterschied: Die XML-Deklaration

Das Allererste, was am Anfang einer XHTML-Datei notiert werden sollte, ist eine so genannte XML-Deklaration. Damit geben Sie an, dass die folgende Datei XML-gerechte Daten enthält. Eine solche Deklaration gibt es in HTML 4.0 nicht, ist also XHTML-spezifisch.

Beispiel für XHTML 1.0:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Notieren Sie diese Angabe so, wie in dem Beispiel gezeigt, in der ersten Zeile einer XHTML-Datei, inklusive der Fragezeichen und Groß-/Kleinschreibung. Derzeit ist beim Attribut `version` nur der Wert `1.0` sinnvoll. Beim Attribut `encoding` können Sie die Zeichenkodierung angeben, nach der die Daten der Datei zu interpretieren sind.

- `encoding="UTF-8"` bedeutet die internationale Zeichenkodierung auf Basis der ISO/IEC-10646-Norm (Unicode), wobei ein Zeichen in der Datei eine variable Breite belegen kann, z.B. ein Byte, aber auch zwei, drei oder vier Bytes.
- `encoding="UTF-16"` bedeutet ebenfalls die internationale Zeichenkodierung auf Basis der ISO/IEC-10646-Norm (Unicode), wobei ein Zeichen in der Datei je nach Unicode-Position entweder zwei oder vier Bytes belegt.

### 14.1.5 Unterschied: Dokumenttyp-Angaben

XHTML 1.0 und HTML 4.0 (siehe Kapitel 3.2.3) haben unterschiedliche Dokumenttyp-Angaben. Die Dokumenttyp-Angabe nimmt Bezug auf die DTD und Sprachversion, die Sie in der Datei verwenden und an die Sie sich halten wollen. Ein strenger Parser kann die Anzeige der Datei z.B. im Browser verhindern, wenn die Datei syntaktische Fehler enthält. Maßgeblich dafür, was ein syntaktischer Fehler ist, ist die DTD, auf die Sie mit der Dokumenttyp-Angabe Bezug nehmen.

Beispiel für HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

Entsprechendes Beispiel für XHTML 1.0:

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

Die Beispiele zeigen, wie Dokumenttypen für HTML 4.01 und XHTML 1.0 für die HTML-Variante »Strict« notiert werden. Auch in XHTML 1.0 gibt es die drei Varianten »Strict«, »Transitional« und »Frameset«, genau wie in HTML 4.0. Die beiden anderen Varianten für XHTML lauten:

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

und

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

**Beachten Sie:** So wie oben notiert, kann ein strenger XML-Parser eine bestehende Internet-Verbindung erfordern und sich die DTD von der angegebenen Web-Adresse laden, um das Dokument gegen die darin formulierten Regeln zu prüfen. Sie können die DTDs in Ihr eigenes Web-Projekt integrieren. Angenommen, Sie haben eine Startdatei namens *index.htm* und die *xhtml1-strict.dtd* in einem eigenen Unterverzeichnis namens *html-dtd* abgelegt, dann können Sie in der *index.htm* notieren:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "html-dtd/xhtml1-strict.dtd">
```

Sie finden die erforderlichen DTDs für XHTML auf der Buch-CD in der HTML-Version dieses Buches zum Download. Die Zip-Datei (31 KByte) enthält die DTDs für alle drei Varianten »Strict«, »Transitional« und »Frameset« sowie drei zusätzliche Dateien mit Definitionen für benannte Zeichen.

### 14.1.6 Unterschied: HTML-Wurzelement mit Namensraumangabe

Das einleitende `<html>`-Tag hat in HTML normalerweise keine Attribute. In XHTML sollten Sie jedoch den Namensraum für XHTML explizit angeben.

Beispiel für HTML 4.01:

```
<html>
<!-- Inhalt der Datei -->
</html>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<!-- Inhalt der Datei -->
</html>
```

Notieren Sie in XHTML im einleitenden `<html>`-Tag immer das Attribut `xmlns=`, das einen XML-Namensraum bezeichnet, und weisen Sie ihm den Wert `http://www.w3.org/1999/xhtml` zu. Dies ist ein spezieller URI, der nicht dazu gedacht ist, eine aufrufbare Web-Adresse anzugeben, sondern lediglich das Schema der Web-Adressierung nutzt, um einen weltweit eindeutigen Namen für den Namensraum zu vergeben. Die Web-Adresse existiert zwar, weil das W3-Konsortium so freundlich war, damit die Häufung unnötiger Aufruffehler auf ihrem Server zu vermeiden – doch die Adresse bedeutet nichts anderes, als wenn man als Name festgelegt hätte: »Superduper-XHTML-Spezifikation«.

### 14.1.7 Unterschied: Strengeres Einhalten des HTML-Grundgerüsts

Wenn Sie in HTML die Elemente `html`, `head` und `body` weglassen, jedoch das Element `title` und ansonsten gültiges HTML notieren, dann ist es aus Sicht von HTML ein fehlerfreies Dokument. In XHTML besteht diese Freiheit nicht. Hier muss eine HTML-Datei zwingend die Form des üblichen Grundgerüsts einhalten und ein `head`- und ein `body`-Element enthalten.

Ein Beispiel für HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<title>Titel</title>
<h1>Text</h1>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Text</title>
<!-- gegebenenfalls andere Elemente im Kopfbereich -->
</head>
<body>
```

```
<h1>Text</h1>
</body>
</html>
```

Zwar ist die gezeigte Form, ein HTML-4.0-Dokument zu notieren, für die Praxis alles andere als empfehlenswert, aber zumindest ist sie nicht verkehrt. Bei einem XHTML-Dokument muss dagegen immer das vollständige Grundgerüst notiert werden.

### 14.1.8 Unterschied: Kleinschreibung

In HTML 4.0 ist es egal, ob Sie `<TABLE BORDER="0">`, `<TABLE border="0">` oder `<Table Border="0">` notieren. HTML unterscheidet bei Namen von HTML-Elementen und Attributnamen nicht zwischen Groß- und Kleinschreibung. Nur bei der Wertzuweisung an manche Attribute wird Groß-/Kleinschreibung unterschieden – aber auch nur im Hinblick etwa auf Script-Sprachen. Bei Attributen mit festen erlaubten Werten, wie etwa `align`, ist es egal, ob Sie `CENTER` oder `center` notieren. Nicht so bei XHTML. XML unterscheidet nämlich strikt zwischen Groß- und Kleinschreibung. Das bedeutet, `<TABLE>` ist etwas anderes als `<table>`. Für XHTML wurde festgelegt, dass **alle Elementnamen und Attributnamen klein geschrieben** werden. Das Gleiche gilt auch für die festen Wertzuweisungen wie `center`.

Ein Beispiel für HTML 4.01:

```
<OBJECT DATA="Video.mpg" TYPE="video/mpeg" ALIGN="LEFT">
</OBJECT>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<object data="Video.mpg" type="video/mpeg" align="left">
</object>
```

Schreiben Sie in XHTML immer alle Elementnamen wie `object` oder Attributnamen wie `data` oder `type` klein. Schreiben Sie Wertzuweisungen im Zweifelsfall ebenfalls immer klein, also beispielsweise bei `align="left"`. Nur dort, wo in einer Wertzuweisung die Großschreibung von bestimmten Buchstaben erforderlich ist, wie im Beispiel beim Dateinamen `Video.mpg`, können Sie den Wert so schreiben, wie es erforderlich ist.

### 14.1.9 Unterschied: Leere Elemente

In HTML 4.0 gibt es diverse leere Elemente. Das sind Elemente ohne Inhalt. Das Abschluss-Tag ist deshalb verboten, weil die Elemente als inhaltsleer definiert sind. Beispiele: `<img>`, `<br>`, `<input>`, `<hr>`. In XML-basierten Sprachen, also auch in XHTML, müssen solche leeren Elemente gesondert gekennzeichnet werden.

Beispiel für HTML 4.01:

```
<p>Text mit<br>Zeilenumbruch</p>
<p></p>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<p>Text mit<br />Zeilenumbruch</p>
<p></p>
```

Notieren Sie unmittelbar vor der schließenden spitzen Klammer des allein stehenden Tags einen Schrägstrich, sodass am Ende die Zeichenfolge `/>` steht. Vor dem Schrägstrich sollte ein Leerzeichen stehen. Alternativ dazu können Sie auch ein Element mit Anfangs- und End-Tag notieren, z.B. `<br></br>`. Dabei darf jedoch nichts, auch kein Leerzeichen und kein Zeilenumbruch, zwischen dem Anfangs- und dem End-Tag stehen, sonst ist es ungültiges XHTML.

Wenn Sie Elemente, die Inhalt haben können, ohne Inhalt verwenden, z.B. `<p></p>`, dann notieren Sie sie in XHTML jedoch stets in dieser Form, und niemals in der Form `<p />`.

#### 14.1.10 Unterschied: Elemente mit optionalem Abschluss-Tag

In HTML 4.0 gibt es diverse Elemente mit optionalem Abschluss-Tag. Beispiele: `<body>`, `<td>`, `<dd>`, `<dt>`, `<option>`. Notieren Sie solche Elemente in XHTML immer mit Anfangs- und Abschluss-Tag.

Ein Beispiel für HTML 4.01:

```
<select name="Auswahl" size="1">
<option>1. Eintrag
<option>2. Eintrag
<option>3. Eintrag
</select>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<select name="Auswahl" size="1">
<option>1. Eintrag</option>
<option>2. Eintrag</option>
<option>3. Eintrag</option>
</select>
```

#### 14.1.11 Unterschied: Allein stehende (leere) Attribute

In HTML 4.0 gibt es diverse Attribute, die keine Wertzuweisung erhalten. Beispiele: `<hr noshade>`, `<td nowrap>`, `<script defer>`. In XML-basierten Sprachen dagegen muss allen Attributen explizit ein Wert zugewiesen werden. Als Regel hat man sich darauf verständigt, als Wert einfach den Attributnamen zuzuordnen. Bei den meisten bekannten Browsern gibt es damit keine Probleme.

Ein Beispiel für HTML 4.01:

```
<td nowrap>Inhalt</td>
<p>Text mit<hr noshade>Trennlinie</p>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<td nowrap="nowrap">Inhalt</td>
<p>Text mit<hr noshade="noshade" />Trennlinie</p>
```

### 14.1.12 Unterschied: Leerraum in Wertzuweisungen

Bei Wertzuweisungen an Attribute müssen Sie in XHTML besser aufpassen als in HTML. Leerzeichen sind erlaubt wo erforderlich, doch Zeilenumbrüche sollten Sie unbedingt vermeiden.

Ein Beispiel für korrektes XHTML 1.0:

```
<p title="Anfang der Geschichte">Text Text Text</p>
```

Beispiel für problematisches XHTML 1.0:

```
<p title="Anfang
der
Geschichte">Text Text Text</p>
```

### 14.1.13 Unterschied: Verweise zu Ankern

In HTML können Sie Verweise zu Ankern notieren, z.B.: `<a href="#Anker">Verweis</a>`. Der entsprechende Anker ist dann notiert mit: `<a name="Anker">irgendwas</a>`. Auch andere Attribute, wie etwa das `usemap`-Attribut im `<img>`-Tag zum Verweis auf ein `map`-Element, bei dem wiederum ein entsprechendes `name`-Attribut notiert ist, verwenden diese Technik.

In XHTML funktionieren solche Verweise nicht. XML benötigt Namen vom Typ »ID«, also dokumentweit eindeutige Bezeichner, um solche Beziehungen auflösen zu können. In XHTML können Sie das Universalattribut `id` verwenden, um solche Verweise zu realisieren. Damit aber auch nicht XHTML-fähige Browser damit zurechtkommen, sollten Sie beide Attribute notieren.

Ein Beispiel für HTML 4.01:

```
<a href="#Anker">Verweis</a>
<p>viel Inhalt</p>
<a name="Anker">irgendwas</a>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<a href="#Anker">Verweis</a>
<p>viel Inhalt</p>
<a id="Anker" name="Anker">irgendwas</a>
```

Notieren Sie in dem `a`-Element für den Anker beide Attribute `id` und `name`. Weisen Sie beiden Attributen exakt den gleichen Namen zu. Das gibt keine Probleme, da beide Attribute unterschiedliche »Namensräume« haben. Ein Browser, der den Verweis nach XHTML-Syntax ausführen will, benötigt das `id`-Attribut, während ein Browser, der den Verweis nach HTML-Syntax ausführt, das `name`-Attribut zur Ermittlung des Ankers verwendet.

#### 14.1.14 Unterschied: Das Universalattribut lang=

XML-basierte Sprachen benutzen normalerweise anstelle von `lang=` das XML-Universalattribut `xml:lang=`. Da das `lang`-Attribut in XHTML jedoch ebenfalls zur Verfügung steht, existieren also zwei Attribute für die gleiche Sache. XHTML-Parser bevorzugen im Zweifelsfall das Attribut `xml:lang=`, das aber von reinen HTML-Parsern nicht erkannt wird. Notieren Sie deshalb in der Praxis stets beide Varianten, falls Sie das Attribut verwenden.

Beispiel für HTML 4.01:

```
<p lang="de">viel Inhalt</p>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<p xml:lang="de" lang="de">viel Inhalt</p>
```

#### 14.1.15 Unterschied: Inhalt von Script- und Style-Bereichen

In HTML 4.0 ist der Inhalt der Elemente `script` und `style` als »CDATA« festgelegt, während er in XHTML als »PCDATA« behandelt wird. Im Klartext bedeutet das, dass bei HTML 4.0 innerhalb eines Script- oder Style-Bereichs HTML-eigene Zeichen wie `<`, `>`, `&` und `"` nicht maskiert werden müssen. Nicht so in XHTML. Wenn Sie beispielsweise innerhalb einer Bedingung vergleichen, ob ein Wert kleiner oder größer ist, als ein anderer Wert müssen Sie die spitzen Klammern außerhalb einer Zeichenkette notieren und können sie an der Stelle deshalb auch nicht maskieren. Um das Problem zu lösen, können Sie den Script-Bereich selber als »CDATA«-Bereich erzwingen. Dazu müssen Sie eine entsprechende XML-typische Notation anwenden. Damit ist das Problem allerdings nur für XML-Parser gelöst. Normale JavaScript-Interpreter etwa meinen, es bei dieser Syntax mit JavaScript zu tun zu haben, und erzeugen einen Fehler, weil es sich nicht um gültiges JavaScript handelt. In diesem Fall bleibt nur, das Script in eine externe Datei auszulagern.

Beispiel für HTML 4.01:

```
<script type="text/javascript">
<!--
  /* JavaScript-Kommentar: jetzt folgt ein kleines Script */
  if(parseInt(navigator.appVersion) < 4)
    alert("Oh, ein sehr alter Browser");
  /* und damit ist das Script auch schon zu ende */
//-->
</script>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<script type="text/javascript">
<!--
<![CDATA[
  /* JavaScript-Kommentar: jetzt folgt ein kleines Script */
  if(parseInt(navigator.appVersion) < 4)
    alert("Oh, ein sehr alter Browser");
  /* und damit ist das Script auch schon zu ende */
```

```
]]>
//-->
</script>
```

Die »CDATA«-Deklaration beginnt mit `<![CDATA[` und endet mit `]]>`. Notieren Sie dies bei XHTML zusätzlich innerhalb des HTML-Kommentars, in den Sie Inhalte von Script- und Style-Bereiche setzen sollten. Andernfalls würde ein XML-Parser das `<`-Zeichen innerhalb des Scripts als Fehler anmeckern.

Um Fehler bei JavaScript-Interpretern zu vermeiden, bleibt die Möglichkeit, JavaScript in separaten Dateien zu notieren. Auch bei CSS Stylesheets besteht die Möglichkeit, Formate zentral in separater CSS-Datei zu definieren.

### 14.1.16 Unterschied: Ausnahmen von Verschachtelungsregeln

In HTML 4.0 sind einige Ausnahmen bei Verschachtelungsregeln für Elemente definiert. So ist beispielsweise festgelegt, dass ein `a`-Element kein anderes `a`-Element enthalten darf. Mit Hilfe von SGML ist die Formulierung solcher Ausnahmen möglich. XML bietet dagegen keine Möglichkeit an, solche Ausnahmen zu formulieren. In den XML-basierten DTDs (Dokumenttyp-Definitionen) von XHTML fehlen solche speziellen Verschachtelungsverbote daher. Da aber eine möglichst 100%ige Kompatibilität zwischen HTML und XHTML erreicht werden sollte, blieb dem W3-Konsortium nur der Ausweg, diese Verschachtelungsverbote verbal als »normativen Anhang« zu den DTDs von XHTML zu erklären.

Folgende Verschachtelungsregeln sind betroffen:

- `a`-Elemente dürfen keine weiteren `a`-Elemente enthalten.
- `pre`-Elemente dürfen keine Elemente `img`, `object`, `big`, `small`, `sub` oder `sup` enthalten.
- `button`-Elemente dürfen keine Elemente `input`, `select`, `textarea`, `label`, `button`, `form`, `fieldset`, `iframe` oder `isindex` enthalten.
- `label`-Elemente dürfen keine weiteren `label`-Elemente enthalten.
- `form`-Elemente dürfen keine weiteren `form`-Elemente enthalten.

### 14.1.17 Beispiel eines XHTML-gerechten Dokuments

Das folgende Beispiel zeigt ein vollständiges XHTML-Dokument, in dem Sie die Unterschiede zu HTML noch einmal im Zusammenhang sehen können. Sie können das zugehörige Anzeigebeispiel einmal als Datei mit der Endung `.htm`, einmal als Datei mit der Endung `.xhtml` und einmal als Datei mit der Endung `.xml` aufrufen, um selber zu testen, was in Ihrem Browser dann passiert. Es handelt sich um exakt den gleichen Inhalt, nur die Dateinamen sind unterschiedlich. Das Beispiel:

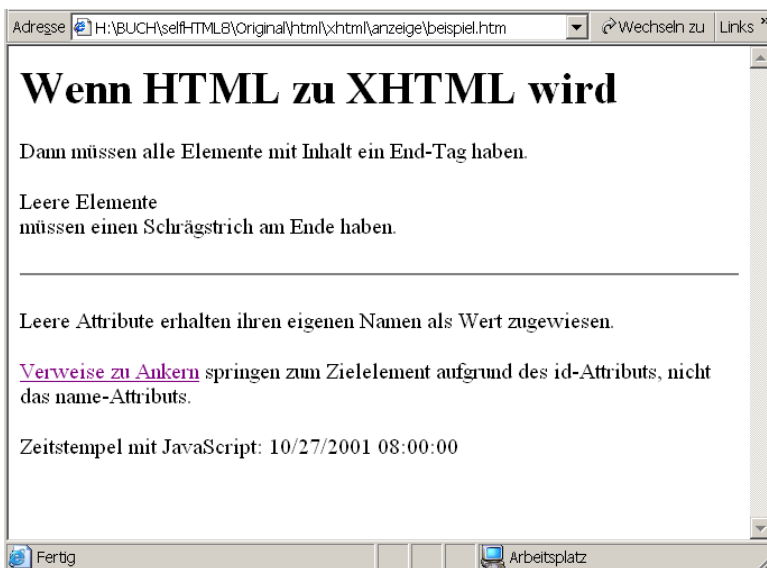
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Wenn HTML zu XHTML wird</title>
```



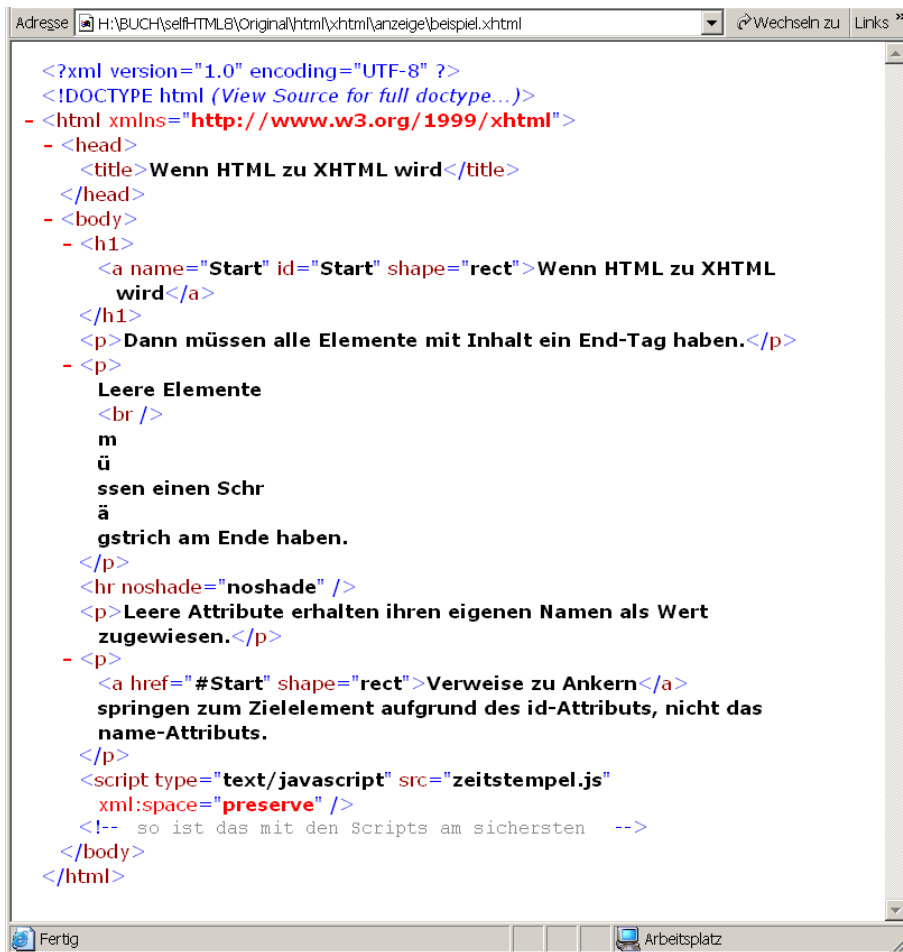
```

</head>
<body>
<h1><a name="Start" id="Start">Wenn HTML zu XHTML wird</a></h1>
<p>Dann müssen alle Elemente mit Inhalt ein End-Tag haben.</p>
<p>Leere Elemente<br />
müssen einen Schrägstrich am Ende haben.</p>
<hr noshade="noshade" />
<p>Leere Attribute erhalten ihren eigenen Namen als Wert zugewiesen.</p>
<p><a href="#Start">Verweise zu Ankern</a> springen zum Zielelement
aufgrund des id-Attributs, nicht das name-Attributs.</p>
<script type="text/javascript" src="zeitstempel.js"></script>
<!-- so ist das mit den Scripts am sichersten -->
</body>
</html>

```



14.1: Die Beispieldatei mit der Erweiterung .htm im Internet Explorer 6.0.



14.2: Dieselbe Datei, nur diesmal mit der Erweiterung .xhtml.

Das Beispiel zeigt ein vollständiges XHTML-Dokument mit verschiedenen typischen Eigenheiten. Die DTD mit dem Namen `xhtml1-strict.dtd` liegt im gleichen Verzeichnis wie das Dokument selber, daher ist bei der Dokumenttyp-Angabe nur `xhtml1-strict.dtd` als URI notiert – ohne weitere Web-Adresse oder Pfadangabe.

## 14.2 Modularisierung von XHTML und XHTML 1.1

### 14.2.1 Allgemeines zur Modularisierung von XHTML

HTML 4.0 und XHTML 1.0 sind – auch wenn sie von unwissenden Schwätzern gerne wegen ihrer Schlichtheit belächelt werden – eigentlich sehr umfangreiche Sprachen mit einem nicht ganz einfachen Regelwerk aus über 80 Elementen, ihren möglichen Attributen, Verschachte-

lungsregeln und einer ganzen Reihe benannter Zeichen. Beide Sprachen bieten damit ein brauchbares Arsenal für Hypertext-Dokumente an. Egal ob einfache Porträt-Homepages oder wissenschaftliche Abhandlungen – der Strukturvorrat von HTML/XHTML ist durchaus reichhaltig und für die meisten Zwecke ausreichend. Allerdings nicht für alle Zwecke.

Der Umfang beider Sprachen und ihre nicht ganz trivialen Regeln stellen andererseits hohe Anforderungen an die auslesende Software und deren Darstellungsfähigkeiten. Von einem modernen Web-Browser, der auf einem modernen PC läuft, kann man diese Fähigkeiten verlangen. Aber von Software, die beispielsweise in einem Handy, einem Pocket-Computer oder einem simplen Sprachwiedergabegerät läuft, kann man nicht unbedingt die gleichen Fähigkeiten erwarten.

HTML/XHTML decken also einerseits trotz ihres beträchtlichen Sprachumfangs nicht alle denkbaren Wünsche ab, und andererseits sind sie für manche Zwecke schlichtweg überdimensioniert. Aus diesem Grund gehen die Überlegungen des W3-Konsortiums in die Richtung, zumindest bei XHTML zu ermöglichen, dass sowohl abgespecktere als auch erweiterte Sprachvarianten davon möglich sind. Handy-Software könnte dann beispielsweise eine abgespecktere Sprachvariante interpretieren, während ein Mathematiker, der seine Arbeit publizieren möchte, XHTML um Elemente aus einer speziellen Auszeichnungssprache für mathematische Formeln erweitern kann.

Das Konzept zum Abspecken und Erweitern von XHTML wird als **Modularisierung** bezeichnet. Dass es nur bei XHTML und nicht bei HTML angewendet werden soll, liegt daran, dass für die Modularisierung XML-typische Techniken angewendet werden sollen. Die Modularisierung stellt nämlich bestimmte formale Regeln zur Verfügung, die abgespeckte oder erweiterte Varianten von XHTML einhalten müssen.

Das W3-Konsortium bietet Beschreibungen an, wie Sie zum Erstellen regelkonformer eigener Module vorgehen müssen. Über die W3-Einstiegsseite zu HTML/XHTML ([www.w3.org/MarkUp/](http://www.w3.org/MarkUp/)) können Sie entsprechende aktuelle Beschreibungen aufrufen. Um die Beschreibungen zu verstehen, benötigen Sie allerdings gute Kenntnisse im Erstellen eigener DTDs.

Neben der Möglichkeit, XHTML für eigene Zwecke zu modularisieren, macht das W3-Konsortium aber auch selber von dieser Möglichkeit Gebrauch. Dazu hat man zunächst die normativen Grundlagen zur Modularisierung geschaffen. Das entsprechende Dokument mit dem Titel *Modularization of XHTML* ([www.w3.org/TR/xhtml-modularization](http://www.w3.org/TR/xhtml-modularization)) liegt als Empfehlung (*Recommendation*) des W3-Konsortiums vor. Ausgehend davon hat man **XHTML 1.1** als modulbasiertes Sprachensystem entworfen. Auch XHTML 1.1 liegt als Empfehlung des W3-Konsortiums vor, unter dem Titel *XHTML 1.1 – Module-based XHTML* ([www.w3.org/TR/xhtml11](http://www.w3.org/TR/xhtml11)). XHTML 1.1 trennt sich endgültig von den in XHTML 1.0 und HTML 4.0 als *deprecated* (missbilligt) eingestuft Elementen und Attributen. Es entspricht nur noch der Sprachvariante »Strict«. Damit fallen zugleich auch die Frames wieder aus dem Konzept.

Inwieweit sich das Konzept der Modularisierung durchsetzen wird, bleibt abzuwarten. Denn wer sich mit dem Design eigener Auszeichnungssprachen befasst, kann schließlich auch direkt die Möglichkeiten von XML nutzen, um entsprechende Sprachen zu entwerfen. Und »einfacher« als XML ist das Konzept der Modularisierung sicher nicht – im Gegenteil: Es setzt im Grunde die Beherrschung von XML voraus und reizt dessen Möglichkeiten aus.

### 14.2.2 Die Module von XHTML 1.1

Die folgende Tabelle listet die Module auf, aus denen XHTML 1.1 besteht. Auf die Praxis als XHTML-Anwender hat diese Modularisierung keinen Einfluss. Sie können Ihre XHTML-Dokumente schreiben wie gewohnt – mit der Einschränkung, dass Sie nichts mehr verwenden dürfen, was noch aus den HTML-Varianten »Transitional« und »Frameset« stammt.

| Name des Moduls       | Elemente                                                                                                            | Erläuterung                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure             | body head html title                                                                                                | Modul für die Elemente, die das Grundgerüst eines XHTML-Dokuments bilden.                                                                                                 |
| Text                  | abbr acronym address<br>blockquote br cite code dfn<br>div em h1 h2 h3 h4 h5 h6 kbd<br>p pre q samp span strong var | Modul für alle Block- und Inline-Elemente, die Text enthalten können und logische Auszeichnungen darstellen, also keinen Hinweis auf ihre visuelle Darstellung enthalten. |
| Hypertext             | a                                                                                                                   | Modul für Anker und Verweise innerhalb des Dokuments.                                                                                                                     |
| List                  | dl dt dd ol ul li                                                                                                   | Modul für nummerierte Listen, Aufzählungslisten und Definitionslisten (Glossarlisten).                                                                                    |
| Object                | object param                                                                                                        | Modul für Multimedia-Referenzen.                                                                                                                                          |
| Presentation          | b big hr i small sub sup tt                                                                                         | Modul für Elemente, die Text enthalten können und physische Auszeichnungen darstellen, also einen Hinweis auf ihre visuelle Darstellung enthalten.                        |
| Edit                  | del ins                                                                                                             | Modul für Änderungsmarkierungen im Text.                                                                                                                                  |
| Bidirectional Text    | bdo                                                                                                                 | Modul für mehrsprachige Dokumente, die Text in Sprachen mit unterschiedlicher Schreibrichtung enthalten.                                                                  |
| Forms                 | button fieldset form input<br>label legend select optgroup<br>option textarea                                       | Modul für Formulare und Formularelemente.                                                                                                                                 |
| Table                 | caption col colgroup table<br>tbody td tfoot th thead tr                                                            | Modul für Tabellen und Tabellenelemente.                                                                                                                                  |
| Image                 | img                                                                                                                 | Modul für Grafikreferenzen.                                                                                                                                               |
| Client-side Image Map | area map                                                                                                            | Modul für Grafiken mit weisseisensitiven Flächen, vom Browser allein darzustellen.                                                                                        |
| Server-side Image Map | ismap (bei img)                                                                                                     | Modul für Grafiken mit weisseisensitiven Flächen, vom Browser in Kommunikation mit dem Web-Server darzustellen.                                                           |

| <i>Name des Moduls</i> | <i>Elemente</i>               | <i>Erläuterung</i>                                                                                                                                                                                                  |
|------------------------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Intrinsic Events       | on* (Event-Handler-Attribute) | Modul für alle Event-Handler, die als Attribut in Elementen notiert werden können und eine Schnittstelle für Script-Sprachen bilden.                                                                                |
| Metainformation        | meta                          | Modul für Meta-Daten für Suchmaschinen und Browser.                                                                                                                                                                 |
| Scripting              | noscript script               | Modul für Script-Bereiche und Bereiche für Browser, die keine Scripts ausführen können.                                                                                                                             |
| Stylesheet             | style (Element)               | Modul für Bereiche, in denen zentrale Stylesheet-Formate definiert werden.                                                                                                                                          |
| Style Attribute        | style (Attribut)              | Modul für das style-Attribut.                                                                                                                                                                                       |
| Link                   | link                          | Modul für logische Beziehungen zu anderen Dateien.                                                                                                                                                                  |
| Base                   | base                          | Modul für Adressbasis und Zielfensterbasis.                                                                                                                                                                         |
| Ruby Annotation        | ruby rbc rtc rb rt rp         | Modul für Ruby-Text. Die entsprechenden Elemente sind neu. Ruby-Text ist eine Textform, die in fernöstlichen Sprachen verwendet wird, um Silben- oder Wortzeichen mit zusätzlichen Bedeutungshinweisen zu versehen. |

# 15 CSS Stylesheets

## 15.1 CSS Stylesheets und HTML

### 15.1.1 Sinn und Zweck von CSS Stylesheets

CSS Stylesheets sind eine unmittelbare Ergänzung zu HTML. Es handelt sich dabei um eine Sprache zur Definition von Formateigenschaften einzelner HTML-Elemente. Mit Hilfe von Stylesheets können Sie beispielsweise bestimmen, dass Überschriften 1. Ordnung eine Schriftgröße von 18 Punkt haben, in der Schriftart Helvetica, aber nicht fett erscheinen, und mit einem Abstand von 1,75 Zentimeter zum darauffolgenden Absatz versehen werden. Angaben dieser Art sind mit reinem HTML nicht möglich.

Das ist aber nur der Anfang. Stylesheets bieten noch viel mehr Möglichkeiten. So können Sie HTML-Elemente – egal ob Textabsätze, Listen, Tabellenzellen oder Formulare – mit einer eigenen Hintergrundfarbe, einem eigenen Hintergrundbild (Wallpaper) oder mit diversen Rahmen ausstatten. Sie können Elemente pixelgenau im Anzeigefenster des WWW-Browsers positionieren. Für Print-Layouts stehen Möglichkeiten zur Definition von Seitenlayout und Textflusskontrolle bereit. Für die akustische Wiedergabe von Web-Seiten gibt es ein ganzes Arsenal an Befehlen, um die künstliche Sprachausgabe feinzusteuern. Einige CSS-Eigenschaften sind auch in der Lage, das Anzeigefenster des Browsers zu beeinflussen, so etwa das Aussehen des Cursors. Spezielle Filter schließlich, die allerdings rein Microsoft-spezifisch sind, erlauben Grafikeffekte bei normalen Texten, die aus Grafikprogrammen wie Photoshop bekannt sind.

Ein weiteres wichtiges Leistungsmerkmal von CSS ist die Möglichkeit, zentrale Formate zu definieren. So können Sie beispielsweise im Kopf einer HTML-Datei zentrale Definitionen zum Aussehen einer Tabellenzelle notieren. Alle Tabellenzellen der entsprechenden HTML-Datei erhalten dann die Formateigenschaften, die einmal zentral definiert sind. Das spart Kodierarbeit und macht die HTML-Dateien kleiner. Sie können Ihre Stylesheet-Definitionen sogar in separaten Dateien notieren. Die Stylesheet-Dateien können Sie in beliebig vielen HTML-Dateien referenzieren. Auf diese Weise entwerfen Sie für große Projekte einheitliche Layouts. Mit ein paar kleinen Änderungen in einer zentralen Stylesheet-Datei können Sie dann für Hunderte von HTML-Dateien ein anderes Aussehen bewirken.

CSS Stylesheets unterstützen also erstens die professionelle Gestaltung beim Web-Design, und zweitens helfen sie beim Corporate Design für große Projekte oder für unternehmensspezifische Layouts.

### 15.1.2 Stylesheet-Sprachen, -Versionen und -Informationen

Genau wie HTML wird auch CSS vom W3-Konsortium normiert. Es handelt sich also um einen firmenunabhängigen, offen dokumentierten und frei verwendbaren Standard. Für CSS gibt es

genau wie für HTML eine Arbeitsgruppe beim W3-Konsortium, die sich um die Weiterentwicklung der Sprache kümmert und sich dabei dem Regelwerk zur Entstehung der so genannten Recommendations (Empfehlungen) des W3-Konsortiums unterzieht.

Wie bei HTML, so gibt es auch bei CSS Sprachversionen, die auf den Neuauflagen der Recommendations basieren. Die Version 1.0 von CSS wurde bereits im Jahr 1996 herausgegeben und 1999 nochmals überarbeitet. 1998 erhielt die Version 2.0 den Status einer Empfehlung. Mittlerweile wird an Version 3.0 gearbeitet. In der vorliegenden Dokumentation werden die beiden ersten Versionen berücksichtigt. Folgende Symbole werden zur Auszeichnung der Sprachversionen verwendet:

**CSS 1.0** Bestandteil der ersten CSS-Version.

**CSS 2.0** Bestandteil der zweiten CSS-Version.

Es gibt mehrere Sprachen zum Definieren von Stylesheets, CSS ist nur eine davon. Aber es ist diejenige Sprache, die speziell als Ergänzung für HTML geschaffen wurde und deshalb für HTML-basierte Web-Seiten optimiert ist. CSS steht für »Cascading Style Sheets«. Andere Style-Sprachen sind die »Document Style Semantics and Specification Language« (DSSSL), die für SGML konzipiert wurde und ebenso wie SGML etwas »sophisticated« ist, und die mittlerweile bekanntere »Extensible Stylesheet Language« (XSL), die als Formatierungssprache für XML entwickelt wurde.

Im Web können Sie die bisherige Entwicklung und die Weiterentwicklung von CSS auf den Seiten des W3-Konsortiums verfolgen:

- Einstiegsseite zu Stylesheets beim W3-Konsortium ([www.w3.org/Style/](http://www.w3.org/Style/)): Über diese Seite gelangen Sie zu Informationen über CSS, XSL und DSSSL sowie über Softwareprojekte rund um Style-Unterstützung.
- Einstiegsseite zu CSS Stylesheets beim W3-Konsortium ([www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)): Über diese Seite gelangen Sie direkt zu aktuellen Nachrichten und Informationen über CSS.

### 15.1.3 Stylesheets, Web-Browser und Anwender

Netscape 4.x interpretiert viele Eigenschaften der CSS-Version 1.0 und einen Teil der Eigenschaften der CSS-Version 2.0. Der MS Internet Explorer kennt weite Teile der CSS-Version 1.0 bereits seit seiner Produktversion 3.0 aus dem Jahr 1996. In der Version 4.0 interpretiert er einen Teil der CSS-Version 2.0 und einige spezielle, von Microsoft eingeführte Eigenschaften. Auf dem Niveau der 4er-Generation beider Browser ist die CSS-Implementierung jedoch noch unvollständig und recht unterschiedlich, und speziell beim 4er-Netscape gibt es zum Teil noch haarsträubende Fehler. Seit der 5er-Version des Internet Explorers und der 6er-Version von Netscape ist die Situation etwas besser geworden. Der Opera-Browser befindet sich in Sachen CSS spätestens seit seiner 5er-Version ebenfalls auf hohem Niveau. Rundum vollständige Implementierungen von CSS 2.0 gibt es in den führenden Browser-Produkten jedoch noch immer nicht.

Das alles entbindet Sie beim Entwerfen von Styles für Ihre Web-Seiten deshalb nicht davon, alle Formate mit verschiedenen Browsern auszutesten. Die CSS-Implementierungen der Browser sind einfach noch nicht so einheitlich, dass Sie darauf vertrauen können, dass alles in jedem Browser genauso aussieht wie bei dem, mit dem Sie bevorzugt arbeiten. Ferner sollte Ihnen be-

wusst sein, dass die optische Wirkung von Web-Seiten auch stark von der eingesetzten Hardware abhängig ist. Web-Seiten werden nun mal nicht nur auf High-End-PCs mit 19-Zoll-Bildschirmen und 16,7 Millionen Farben gelesen. Sie werden auch mit Sub-Notebooks und Handheld-Computern gelesen und mit veralteten PCs angezeigt. Gerade die unterschiedlichen Technologien, die heute im Bereich der Bildschirme auf dem Markt sind, sorgen für sehr unterschiedliche Wirkungen etwa von Farben. Was auf dem einen Bildschirm als beige wirkt, sieht auf dem nächsten aus wie schweinchenrosa. Zwar lassen sich Bildschirme kalibrieren, doch Sie können ja schlecht vom Anwender verlangen, dass er vor dem Betrachten Ihrer Web-Seiten erst mal seinen Bildschirm neu einstellt. Immerhin besteht ab CSS 2.0 die Möglichkeit, unterschiedliche CSS-Formate für unterschiedliche Ausgabemedien zu definieren. Doch alle Probleme lassen sich damit auch nicht beseitigen.

Bedauerlich ist weiterhin, dass Schriftgrößenangaben in CSS am Bildschirm leider nicht zu einheitlichen Ergebnissen führen. Während im Print-Bereich eine 10-Punkt-Schrift immer gleich groß ist, weil nun mal definiert ist, wie groß ein Punkt ist, werden solche Angaben am Bildschirm in Pixel umgerechnet. Dabei ist aber wiederum nicht festgelegt, wie groß ein Pixel ist. Und nicht mal die Umrechnungsfaktoren sind die gleichen. So rechnet etwa Windows Schriftgrößenangaben anders in Pixel um als etwa grafische Oberflächen unter Linux oder als das Macintosh-Betriebssystem. Kurzum – vieles in CSS steht nur genau in der Datei, aber die Ergebnisse am Bildschirm können doch recht unterschiedlich und nicht selten unbefriedigend sein.

Ein ganz wichtiger Aspekt beim Entwerfen von Styles für Web-Seiten sind aber auch die Menschen, die diese Web-Seiten als Besucher betrachten sollen. Es gibt nicht wenige Menschen, die Farben und Formen aufgrund visueller Schwächen nur eingeschränkt wahrnehmen können. Was Sie beim Designen vielleicht als »edel« empfinden, empfinden solche Menschen nur noch als »unleserlich«. Und »unleserlich« bedeutet: Sie haben wieder einen Besucher weniger auf Ihren Seiten. Stylesheets verführen aufgrund ihrer Möglichkeiten leicht zur Verwendung ungewöhnlicher Schriftarten oder zu Dingen wie Ton-in-Ton-Farbgebung. So »schön« das auch aussehen mag – es kann je nach Anzeigegerät und visuellem Auflösungsvermögen beim Betrachter schnell auf Kosten der »Usability«, also der Zweckmäßigkeit gehen. Andererseits können Sie Stylesheets auch explizit dafür nutzen, um Seiten für sehschwache Menschen zu optimieren. Ordentliche Farbkontraste, Schriftarten im normalgroßen Bereich, übliche Schriftweiten und Zeilenhöhen, Vermeidung von kritischen Farbkombinationen wie rot/grün bei Hintergrund/Vordergrund – das sind Eigenschaften, die gut besuchte Web-Seiten aufweisen sollten.

Wenn Sie Web-Seiten gestalten, müssen Sie daraus Ihre eigenen Konsequenzen ziehen. Stylesheets erlauben Ihnen völlig neue Möglichkeiten beim Web-Seiten-Layout. Je intensiver Sie Ihre Layouts jedoch von Stylesheet-Effekten abhängig machen, desto problematischer können die Seiten auf schwacher Hardware oder bei sehschwachen Menschen wirken. Zwar gibt es außer textbasierten Browsern wie Lynx kaum noch Browser mit nennenswerter Verbreitung, die Stylesheets gar nicht unterstützen. Doch bei frühen Varianten von Netscape 4.x, die durchaus noch verbreitet sind, ist leider eine verhängnisvolle Abhängigkeit einprogrammiert: Wenn der Anwender dort JavaScript deaktiviert, zeigt der Browser auch keine CSS Stylesheets an. Der Grund dafür ist, dass im Netscape-Browser ursprünglich mal CSS direkt vom JavaScript-Interpreter interpretiert wurde, weshalb es in Netscape 4.x auch eine heute veraltete Zwitter-Syntax namens JSSS gibt, die CSS-Definitionen mittels JavaScript ermöglicht. Einen Vorteil hat diese Spezialität von Netscape 4.x andererseits: Wenn Sie dort JavaScript deaktivieren, können Sie Ihre CSS-gestylten Web-Seiten mal ohne CSS-Unterstützung testen. Bei jüngeren Versionen der 4.x-



Serie von Netscape lässt sich CSS auch unabhängig von JavaScript deaktivieren. Zu diesem »Härtetest« sollten Sie sich auf jeden Fall zwingen.

### 15.1.4 CSS-Formate und CSS-Eigenschaften

CSS Stylesheets bestehen aus Formaten, die Sie für bestimmte HTML-Elemente oder für eine bestimmte Auswahl von HTML-Elementen definieren. Zum Auswählen einer bestimmten Gruppe von HTML-Elementen, für die ein Format gelten soll, gibt es in CSS die Syntax der sogenannten Selektoren. CSS-Formate lassen sich entweder in einem zentralen Style-Bereich definieren, in einer externen CSS-Datei oder direkt im einleitenden Tag eines HTML-Elements.

CSS-Formate bestehen aus einer oder mehreren Eigenschaften und Wertzuweisungen an diese Eigenschaften. So können Sie beispielsweise ein Format für Überschriften 3. Ordnung definieren, bei dem Sie für Eigenschaften wie Schriftgröße, Schriftfarbe und Absatznachabstand entsprechende Werte bestimmen.

Im folgenden Kapitel 15.2 wird beschrieben, an welchen Stellen Sie CSS in HTML einbinden können, wie die Selektion von Gruppen von HTML-Elementen funktioniert und welche generellen Regeln für Formatdefinitionen gelten. Das Kapitel 16 zu *CSS-Eigenschaften* beschreibt dagegen systematisch die einzelnen möglichen Eigenschaften für die Elementformatierung, die Sie innerhalb von Formaten notieren können, und mögliche Wertzuweisungen für diese Eigenschaften.

## 15.2 CSS Stylesheets in HTML einbinden

### 15.2.1 Formate zentral für eine HTML-Datei definieren

Sie können innerhalb einer HTML-Datei einen Bereich für CSS-Formate definieren. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
/* ... Hier werden die Formate definiert ... */
-->
</style>
</head>
<body>
</body>
</html>
```

Mit `<style...> ... </style>` notieren Sie einen Bereich für Stylesheet-Formatdefinitionen (*style* = *Stil*, *Format*). Im einleitenden `<style>`-Tag müssen Sie den Mime-Type der Stylesheet-Sprache angeben. Für CSS ist das die Angabe `type="text/css"`. Zwischen dem einleitenden Tag und dem abschließenden `</style>` können Sie dann zentrale CSS-Formate definieren. Lesen Sie dazu den Abschnitt 15.3 *Zentrale Formate definieren*.

**Beachten Sie:** Damit ältere Web-Browser, die keine Stylesheets kennen, den Inhalt des `style`-Elements nicht irrtümlich als anzuzeigenden Text interpretieren, können Sie den Inhalt in einen HTML-Kommentar (`<!-- ... -->`) einbinden, so wie im obigen Beispiel. Formate, die Sie auf diese Weise definieren, sind für diese eine HTML-Datei gültig. Es ist durchaus erlaubt, mehrere `style`-Bereiche innerhalb des HTML-Dateikopfs zu notieren. Sinnvoll ist das beispielsweise, wenn Sie mal Style-Definitionen mit unterschiedlichen Style-Sprachen und entsprechend unterschiedlichem Mime-Type in einer HTML-Datei notieren möchten.

### 15.2.2 Formate zentral in separater CSS-Datei definieren

In vielen Fällen werden Sie einheitliche Formate für alle HTML-Dateien Ihres Projekts haben wollen. In diesem Fall brauchen Sie die Angaben nicht in jeder Datei zu wiederholen. Stattdessen können Sie die Formate in einer separaten Textdatei definieren und diese Datei einfach in jeder gewünschten HTML-Datei einbinden. Wenn Sie die Angaben in der separaten Datei ändern, wirken sich die Änderungen einheitlich auf alle Dateien aus, in denen die separate CSS-Datei eingebunden ist. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<link rel="stylesheet" type="text/css" href="formate.css">
<style type="text/css">
<!--
... hier sind datei-spezifische Formate erlaubt ...
-->
</style>
</head>
<body>
</body>
</html>
```

Im Dateikopf einer HTML-Datei können Sie mit `<link...>` eine CSS-Datei referenzieren, die CSS-Formatdefinitionen enthält (*link* = *Verweis*). Innerhalb des `<link>`-Tags müssen die Angaben `rel="stylesheet" type="text/css"` stehen (*rel* = *relation* = *Bezug*, *type* = *Mime-Typ*). Beim Attribut `href` geben Sie die gewünschte Datei an (*href* = *hyper reference* = *Hyper-text-Referenz*). Wenn sich die CSS-Datei in einem anderen Verzeichnis oder auf einem anderen Server befindet, müssen Sie an dieser Stelle Pfadangaben oder absolute URIs notieren. Das funktioniert genauso wie etwa beim Einbinden von Grafiken.

Bei der referenzierten Datei muss es sich um eine reine Textdatei handeln, die die Endung `.css` haben sollte. Die Datei darf nichts anderes als Definitionen zentraler Formate und Kommentare enthalten. Eine solche CSS-Datei können Sie mit jedem einfachen Texteditor erstellen. In der HTML-Version dieses Buches (SELFHTML) wird beispielsweise eine separate CSS-Datei für alle Dokumentdateien verwendet. Die CSS-Datei heißt `selfhtml.css`.

**Beachten Sie:** Wenn Sie mit `<link...>` eine Datei mit CSS-Formatdefinitionen referenzieren, brauchen Sie keinen Bereich `<style...>...</style>`. Im obigen Beispiel wird dennoch ein solcher Bereich definiert. Das soll zeigen, dass Sie beide Arten, Formate zu definieren, miteinander kombinieren können. Wenn Sie beide Arten benutzen, haben Formate, die direkt innerhalb von

`<style...>...</style>` definiert werden, im Konfliktfall Vorrang vor den referenzierten Formaten. So können Sie etwa immer wieder verwendete Formate importieren und einige davon mit dateispezifischen Formaten ergänzen oder überschreiben.

### 15.2.3 Formate innerhalb eines HTML-Elements definieren

Unabhängig davon, ob Sie Formate zentral definieren oder aus einer separaten CSS-Datei einbinden, können Sie einzelne HTML-Elemente formatieren. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<link rel="stylesheet" type="text/css" href="formate.css">
<style type="text/css">
<!--
... hier sind datei-spezifische Formate erlaubt ...
-->
</style>
</head>
<body>
<h1 style="[element-spezifische Formate]">...</h1>
</body>
</html>
```

Mit dem Universalattribut `style=` im einleitenden Tag eines Elements können Sie CSS-Formate für dieses eine Element notieren. Weitere Einzelheiten dazu werden im Abschnitt 15.4 *HTML-Elemente direkt formatieren* beschrieben.

**Beachten Sie:** HTML-Elemente müssen HTML-4.0 konform mit Start- und End-Tag notiert sein, damit CSS-Formatierungen darin angezeigt werden. Dies gilt vor allem für Elemente, bei denen früher mal nur die Notation des Start-Tags genügte, etwa bei `p`, `li`, `option`, `td` und `th`.

Das obige Beispiel zeigt, dass Sie parallel zu den Formatdefinitionen im HTML-Element auch die Methoden zur zentralen Formatdefinition verwenden können. Im Konfliktfall haben stets die »inneren« Formatdefinitionen Vorrang vor den »äußeren«. Wenn Sie also im einleitenden `<h1>`-Tag eine andere Schriftart angeben, als Sie im zentralen `style`-Bereich für `h1`-Elemente festlegen, dann hat die Definition im einleitenden `<h1>`-Tag Vorrang vor der zentralen Definition, wobei, wie diese wiederum Vorrang vor den Definitionen der eingebundenen externen CSS-Datei hat.

### 15.2.4 Verschiedene separate Stylesheets für unterschiedliche Ausgabemedien

Bildschirm und Drucker beispielsweise sind sehr unterschiedliche Ausgabemedien für ansprechend gestaltete Daten. Beide Ausgabemedien haben ihre eigenen Gesetze. Während am Bildschirm etwa helle Schriften auf dunklen Hintergründen attraktiv aussehen, ist das für die Ausgabe am Drucker keine gute Lösung. Bei der Druckerausgabe sehen dagegen Absatzzeinzüge von

mindestens 2 cm besser aus, während durch entsprechende Angaben am Bildschirm möglicherweise kostbarer Präsentationsraum verschenkt wird. Deshalb können Sie für verschiedene Ausgabemedien verschiedene externe CSS-Dateien einbinden, die unterschiedliche Formatdefinitionen enthalten können. Die Software muss beim Präsentieren der Seiten entscheiden, welche der eingebundenen CSS-Dateien maßgeblich ist. Wenn der Web-Browser also beispielsweise die Seiten am Bildschirm anzeigt, sollte er die CSS-Datei verwenden, die Sie explizit für das Medium »Bildschirm« bestimmen, und wenn der Anwender eine Seite ausdruckt, sollte der Browser stattdessen die CSS-Datei verwenden, die Sie für das Medium »Drucker« angeben. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<link rel="stylesheet" media="screen" href="website.css">
<link rel="stylesheet" media="print, embossed" href="druck.css">
<link rel="stylesheet" media="aural" href="speaker.css">
</head>
<body>
</body>
</html>
```

Zunächst benötigen Sie für jedes Ausgabemedium eine Textdatei mit der Endung *.css*, die nichts anderes als die gewünschten CSS-Formatdefinitionen für das gewünschte Ausgabemedium enthält. In einer HTML-Datei können Sie alle CSS-Dateien wie im obigen Beispiel einbinden. Das Einbinden funktioniert genauso wie im Kapitel 15.2.2 *Formate zentral in separater CSS-Datei definieren* beschrieben. Neu ist hierbei nur, dass gleich mehrere `<link>`-Tags zum Einbinden notiert sind und dass diese zusätzlich das Attribut `media=` enthalten. Mit diesem Attribut bestimmen Sie, für welches Ausgabemedium die Datei verwendet werden soll, die Sie dann bei der Angabe `href=` einbinden. Sie können einen oder mehrere Medientypen bei `media=` angeben. Mehrere Angaben werden durch Kommata getrennt. In CSS 2.0 werden folgende möglichen Angaben genannt:

| Angabe                        | Bedeutung                                                                                                                                                                                                                                                                     |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>media="all"</code>      | CSS-Datei gilt für alle Medientypen.                                                                                                                                                                                                                                          |
| <code>media="aural"</code>    | CSS-Datei gilt für computergesteuerte, synthetische Sprachausgabe. Beachten Sie in diesem Zusammenhang auch die <i>CSS-Eigenschaften für Sprachausgabe</i> .                                                                                                                  |
| <code>media="braille"</code>  | CSS-Datei gilt für Ausgabegeräte mit so genannter »Braille-Zeile«. Dabei wird Text eine veränderbare Oberflächenstruktur des Materials der Braille-Zeile umgewandelt und über Abtasten mit dem Finger gelesen. Diese Form des Ausgabemediums ist für blinde Menschen gedacht. |
| <code>media="embossed"</code> | CSS-Datei gilt für Braille-Drucker. Dabei wird Text in Form von abtastbaren Oberflächenstrukturen in Papier oder ein vergleichbares Material gestanzt. Wie <i>braille</i> ist <i>embossed</i> für blinde Menschen gedacht.                                                    |
| <code>media="handheld"</code> | CSS-Datei gilt für die Anzeige auf tragbaren Kleinstcomputern. Web-fähige elektronische Organizer sind typische Vertreter dieser Gattung.                                                                                                                                     |
| <code>media="print"</code>    | CSS-Datei gilt für den Ausdruck auf Papier. Web-Browser sollten diese Formatdefinitionen verwenden, wenn der Anwender die Web-Seite ausdrucken möchte. Beachten Sie dazu auch die <i>CSS-Eigenschaften für Printmedien</i> .                                                  |

| Angabe             | Bedeutung                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| media="projection" | CSS-Datei gilt für die Datenprojektion mit Beamern und ähnlichen Geräten.                                                                                                                    |
| media="screen"     | CSS-Datei gilt für die Bildschirmanzeige.                                                                                                                                                    |
| media="tty"        | CSS-Datei gilt für nichtgrafische Ausgabemedien mit fixer Zeichenbreite, beispielsweise Fernschreiber. Aber auch für textorientierte Browser wie Lynx ist dieser Medientyp interessant.      |
| media="tv"         | CSS-Datei gilt für TV-ähnliche Ausgabegeräte, die sich durch grobe Bildschirm-auflösung und mangelnde Unterstützung beim Scrollen in Text, dafür aber durch Sound-Unterstützung auszeichnen. |

**Beachten Sie:** Der MS Internet Explorer interpretiert diese Angaben ab der Version 4.0 zu einem gewissen Teil. Netscape findet auch in Version 6.0 zwar die richtige Stylesheet-Datei für die Bildschirmformate, ignoriert jedoch beispielsweise Stylesheet-Dateien mit Formaten für den Druck.

Die Syntax mit `<link...>` zum Einbinden separater Stylesheet-Dateien für unterschiedliche Ausgabemedien ist HTML-Syntax. Daneben gibt es auch noch eine CSS-Syntax, die den gleichen Zweck hat. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
@import url(druck.css) print, embossed;
@import url(pocketcomputer.css) handheld;
@import url(normal.css) screen;
-->
</style>
</head>
<body>
</body>
</html>
```

Eine CSS-spezifische Einbindung wird innerhalb eines `style`-Bereichs im Dateikopf notiert. Mit `@import url` starten Sie die Angabe. In Klammern notieren Sie dahinter den Dateinamen bzw. die Adresse der separaten CSS-Datei. Hinter der Dateiangabe notieren Sie, durch Leerraum getrennt, das gewünschte Ausgabemedium. Es dürfen auch mehrere sein, getrennt durch Kommata. Dabei sind die gleichen Medientypen erlaubt wie im HTML-Attribut `media=`. Am Ende der `@import`-Angabe muss ein Strichpunkt stehen.

## 15.2.5 Formatdefinitionen für unterschiedliche Ausgabemedien

Neben der Möglichkeit, Verschiedene separate Stylesheets für unterschiedliche Ausgabemedien (Kapitel 15.2.4) in HTML einzubinden, steht auch die Möglichkeit zur Verfügung, innerhalb eines `style`-Bereichs zwischen unterschiedlichen Ausgabemedien zu unterscheiden. Ein Beispiel:

```

<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
@media print
{
    /* ... Formatdefinitionen zum Drucken ... */
}
@media screen, handheld
{
    /* ... Formatdefinitionen zur Bildschirmausgabe ... */
}
-->
</style>
</head>
<body>
</body>
</html>

```

Mit `@media` definieren Sie innerhalb eines `style`-Bereichs einen Bereich für Formatdefinitionen für ein bestimmtes Ausgabemedium. Für jede Mediensorte, die Sie unterscheiden wollen, können Sie einen solchen Befehl notieren. Hinter dem Befehl `@media` folgt, durch Leerraum getrennt, das gewünschte Ausgabemedium. Sie können auch mehrere Ausgabemedien durch Kommata getrennt in Folge notieren. Erlaubte Angaben sind die gleichen wie im HTML-Attribut `media=` (siehe Kapitel 15.2.4).

Im Anschluss an die `@media`-Angabe müssen Sie **geschweifte Klammern** `{` und `}` notieren. Alle Formatdefinitionen, die Sie innerhalb dieser geschweiften Klammern notieren, sind dann nur für das oder die angegebenen Medien gültig.

### 15.2.6 Kommentare innerhalb von Stylesheet-Bereichen

Um Ihre Formatdefinitionen intern auszukommentieren, steht eine spezielle, an die Programmiersprache C angelehnte Syntax zur Verfügung. Ein Beispiel:

```

<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
    p { color:blue; } /* Format für blauen Text, kreiert am 19.7.2001 */
-->
</style>
</head>
<body>
</body>
</html>

```

Innerhalb von `<style...>...</style>` können Sie mit `/*` – also einem Schrägstrich und einem Stern-Zeichen in Folge – einen Kommentar einleiten. Mit der umgekehrten Zeichenfolge – also mit `*/` – beenden Sie den Kommentar. Anfang und Ende des Kommentars müssen nicht in der gleichen Zeile stehen. Kommentare dürfen jedoch nicht verschachtelt werden.

### 15.2.7 Stylesheet-Sprache ausdrücklich festlegen

Sie können dem Browser in einer HTML-Datei mit Hilfe einer Meta-Angabe ausdrücklich mitteilen, dass diese Datei Stylesheet-Definitionen in einer bestimmten Definitionssprache enthält. Die Angabe ist in der gegenwärtigen Praxis und für die CSS-Sprache nicht zwingend erforderlich. Die Angabe dient dazu, um dem Client (also etwa dem Web-Browser) und aber auch dem Web-Server einen Befehl zur Verfügung zu stellen, der beiden Seiten helfen soll, sich vor dem Interpretieren der Datei über die zu verwendende Stylesheet-Syntax zu einigen. Ein Beispiel:

```
<head>
<meta http-equiv="Content-Style-Type" content="text/css">
</head>
```

Mit `<meta http-equiv="Content-Style-Type" content="text/css">` im Kopf einer HTML-Datei erzwingen Sie die CSS-Syntax als verbindlich für Formatdefinitionen innerhalb dieser Datei. Wenn Sie eine andere Stylesheet-Sprache als CSS verwenden wollen, müssen Sie anstelle von `"text/css"` den Mime-Type der betreffenden Formatdefinitionssprache angeben.

## 15.3 Zentrale Formate definieren

### 15.3.1 Aufbau zentraler Formate

Die hier beschriebenen Möglichkeiten gelten, wenn Sie:

- Formate zentral für eine HTML-Datei definieren oder
- Formate zentral in separater CSS-Datei definieren.

Zentrale Formate folgen einem bestimmten Grundmuster. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
h1 { color:red; font-size:36pt; }
-->
</style>
</head>
<body>
<h1>36 Punkt und Rot!</h1>
</body>
</html>
```

Das Beispiel enthält in einem `style`-Bereich eine Formatdefinition für `h1`-Elemente der Datei, also für Überschriften 1. Ordnung. Solche Formatdefinitionen haben immer den gleichen Aufbau und bestehen aus folgenden Teilen:

- **Selektor:** Als Selektor wird das bezeichnet, was vor den geschweiften Klammern steht. Der Selektor wählt aus, wofür die folgenden Definitionen gelten sollen. Im obigen Beispiel gelten die Formate für alle Überschriften 1. Ordnung (`h1`-Elemente). Es sind jedoch auch komplexere Selektoren möglich.
- **Definitionen:** Die eigentlichen Definitionen zum Format stehen stets in geschweiften Klammern `{` und `}`. Sie bestehen darin, dass eine oder mehrere CSS-Eigenschaften notiert werden und einen Wert erhalten. Im obigen Beispiel werden etwa die CSS-Eigenschaften `color` (Schriftfarbe) und `font-size` (Schriftgröße) verwendet. Der Eigenschaft `color` wird der Wert `red` zugewiesen und der Eigenschaft `font-size` der Wert `36pt`. Zwischen Eigenschaft und Wertzuweisung muss stets ein Doppelpunkt stehen. Abgeschlossen wird eine Definition mit einem Strichpunkt `;`. Nur bei der letzten Definition vor der schließenden geschweiften Klammer darf der Strichpunkt auch entfallen. Zu den einzelnen Eigenschaften und ihren möglichen Werten siehe Kapitel 16 über *CSS-Eigenschaften*.

**Beachten Sie:** Zwar sind innerhalb einer Definition durchaus Leerzeichen erlaubt, aber frühe Versionen von Netscape 4.x haben sich dabei als sehr empfindlich erwiesen und ignorieren solche Definitionen. Obwohl es nicht so gut lesbar ist, ist es deshalb besser, keine Leerzeichen innerhalb einer Definition zu verwenden. So werden Angaben wie `color:red; font-size:36pt;` in jedem Fall akzeptiert, aber `color: red; font-size: 36pt;` können bei einigen älteren Netscape-Versionen zu Fehlern führen.

### 15.3.2 Formate für HTML-Elemente definieren

In einem `style`-Bereich oder in einer externen CSS-Datei können Sie für HTML-Elementtypen, zum Beispiel zu Absatztypen und Textgestaltung oder für Tabellen-Elementtypen, CSS-Formate definieren. Wenn Sie entsprechende HTML-Elemente dann in der HTML-Datei verwenden, werden alle Formate angewendet, die Sie für den Elementtyp definiert haben. So können Sie beispielsweise für Überschriften 1. Ordnung, definieren, dass diese in der Schriftart Helvetica, 20 Punkt, rot, fett und kursiv mit einem Absatznachabstand von 16 Punkt angezeigt werden. Wenn Sie dann im Text der HTML-Datei eine Überschrift 1. Ordnung wie gewohnt definieren, werden diese Formateigenschaften berücksichtigt. Ein Beispiel:

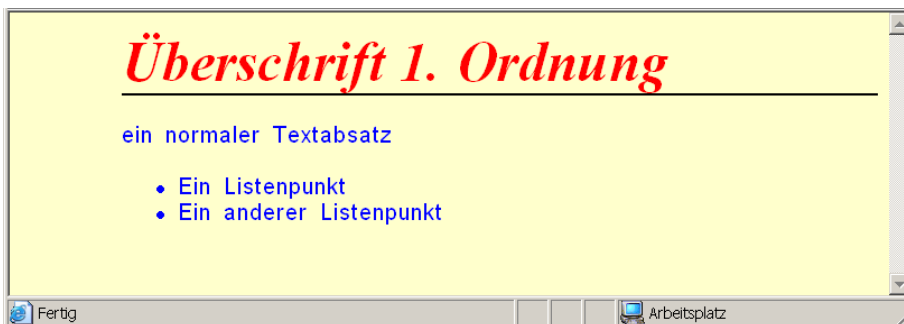
```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
body { background-color:#FFFFCC;
      margin-left:100px; }
h1 { font-size:48pt;
     color:#FF0000;
     font-style:italic;
     border-bottom:solid thin black; }
p,li { font-size:12pt;
```



```

    line-height:14pt;
    font-family:Helvetica,Arial,sans-serif;
    letter-spacing:0.2mm;
    word-spacing:0.8mm;
    color:blue; }
-->
</style>
</head>
<body>
<h1>Überschrift 1. Ordnung</h1>
<p>ein normaler Textabsatz</p>
<ul>
<li>Ein Listenpunkt</li>
<li>Ein anderer Listenpunkt</li>
</ul>
</body>
</html>

```



15.1: Alle Überschriften, Listenpunkte und Absätze haben besondere CSS-Formate erhalten.

Um ein zentrales Format für alle HTML-Elemente eines bestimmten Typs zu definieren, notieren Sie als Selektor den Namen des HTML-Elementtyps, und zwar ohne spitze Klammern. Im obigen Beispiel werden `body` (Dokument), `h1` (Überschriften 1. Ordnung), `p` (Textabsätze) und `li` (Listeneinträge) auf diese Weise notiert. Wenn Sie ein Format für mehrere HTML-Elementtypen definieren wollen, geben Sie alle gewünschten Elementtypen an und trennen sie durch Kommata, so wie im obigen Beispiel `p,li`.

Es bedeutet also das Gleiche, wenn Sie notieren:

```

h1 { font-family:Helvetica,sans-serif; }
h2 { font-family:Helvetica,sans-serif; }

```

oder wenn Sie notieren:

```

h1, h2 { font-family:Helvetica,sans-serif; }

```

Dahinter folgen die gewünschten Definitionen. Im obigen Beispiel wird dem `body`-Element eine hellgelbe Hintergrundfarbe (`background-color:#FFFFCC;`) und ein linker Randabstand von 100 Pixeln (`margin-left:100px;`) zugewiesen. Überschriften 1. Ordnung (`h1`) erhalten eine

Schriftgröße von 48 Punkt (`font-size:48pt;`), die Schriftfarbe rot (`color:#FF0000;`) und den Schriftstil kursiv (`font-style:italic;`). Textabsätze (`p`) und Listenelemente (`li`) sollen in 12 Punkt Schriftgröße (`font-size:12pt;`) mit einer Zeilenhöhe von 14 Punkt (`line-height:14pt;`) in Schriftart Helvetica oder, wenn nicht verfügbar, in Arial, und wenn ebenfalls nicht verfügbar, in irgendeiner Sans-Serif-Schrift dargestellt werden (`font-family:Helvetica,Arial,sans-serif;`). Außerdem werden auch noch Angaben zum Zeichenabstand (`letter-spacing`), Wortabstand (`word-spacing`) und zur Textfarbe gemacht.

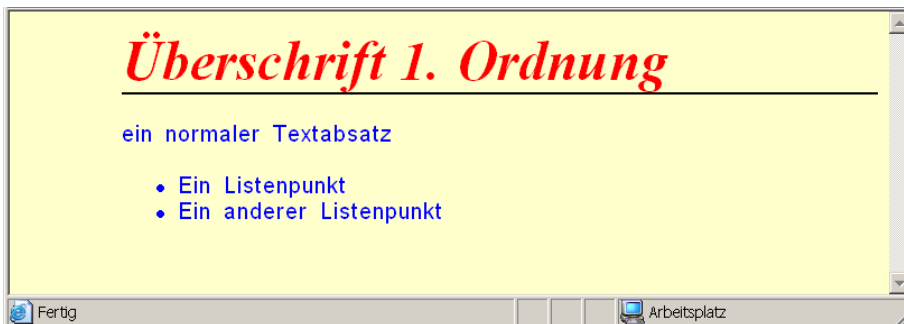
### 15.3.3 Formate für verschachtelte HTML-Elemente definieren

Wenn Sie nichts anderes angeben, übernimmt ein HTML-Element, das innerhalb eines anderen HTML-Elements vorkommt, dessen Eigenschaften und fügt seine eigenen Eigenschaften nur hinzu. Wenn Sie beispielsweise für Überschriften 1. Ordnung die Schriftart Times und die Farbe Rot definieren, erscheint Text, der innerhalb einer solchen Überschrift mit `<i>...</i>` formatiert wird, ebenfalls rot und in Times, aber zusätzlich kursiv.

Sie können mit Hilfe von Stylesheets jedoch bestimmen, dass ein HTML-Element bestimmte Eigenschaften nur dann hat, wenn es innerhalb eines bestimmten anderen HTML-Elements vorkommt. So können Sie etwa bestimmen, dass `<i>...</i>` innerhalb von Überschriften nicht kursiv, stattdessen aber in blauer Farbe dargestellt wird, während das gleiche Element innerhalb anderer HTML-Tags nach wie vor nichts anderes als eine kursive Darstellung bewirkt. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
h1 { color:red; }
h1 i { color:blue; font-style:normal; }
-->
</style>
</head>
<body>
<h1>Wir lernen <i>Styleheets</i></h1>
<p>Wir lernen <i>Stylesheets</i></p>
</body>
</html>
```

Im Beispiel wird festgelegt, dass Textabschnitte, die mit `<i> ... </i>` ausgezeichnet sind, nicht wie sonst üblich kursiv, sondern normal (`font-style:normal;`), stattdessen aber mit blauer Farbe (`color:blue;`) dargestellt werden – aber nur dann, wenn das `i`-Element innerhalb einer Überschrift 1. Ordnung (`h1`) vorkommt. Dazu notieren Sie zuerst den Namen des übergeordneten Elementtyps, im Beispiel `h1`, und dahinter, durch Leerraum getrennt, den Namen des inneren Elementtyps, im Beispiel `i`.



15.2: Ein Format für verschachtelte HTML-Elemente wirkt nur in der passenden »Umgebung«.

In der ersten Zeile innerhalb des `<body>`-Bereichs im obigen Beispiel kommt diese spezielle Definition zum Tragen. In der zweiten Zeile dagegen, wo `<i>...</i>` innerhalb eines Textabsatzes vorkommt, hat es die übliche Wirkung.



### Weitere Möglichkeiten für verschachtelte HTML-Elemente bei CSS 2.0

Ab CSS 2.0 gibt es weitere Möglichkeiten für Selektoren-Angaben zu verschachtelten Elementen. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
div i { color:red; }
div > p { color:blue; }
div * b { color:violet; }
div + p { margin-top:100px; }
-->
</style>
</head>
<body>
<div align="center">
Text in einem div-Bereich mit <b>nur fettem Text</b> und
<i>kursivem und <b>dazu fettem</b> Text</i>.
<p>Textabsatz in einem div-Bereich mit <b>nur fettem Text</b> und <i>kursivem
und <b>dazu fettem</b> Text</i>.</p>
<table border="1"><tr>
<td><p>Textabsatz in einer Tabellenzelle.</p></td>
</tr></table>
</div>
<p>Textabsatz ausserhalb eines div-Bereichs mit <b>nur fettem Text</b>
und <i>kursivem und <b>dazu fettem</b> Text</i>.</p>
<p>Noch ein Textabsatz.</p>
</body>
</html>
```

Die Selektoren im Beispiel haben folgende Bedeutung:

- `div i { color:red; }`  
Damit wird wie nach CSS 1.0-Syntax bekannt festgelegt, dass *i*-Elemente, sofern sie innerhalb eines `div`-Bereichs vorkommen, rote Textfarbe (`color:red;`) erhalten. Dabei wird das *i*-Element aber auch dann rot formatiert, wenn es z.B. in dieser Form vorkommt: `<div><ul><li><i></i></li></ul></div>`. Deshalb gibt es ab CSS 2.0 die Möglichkeit, genauer festzulegen, für welche Verschachtelung die Formatdefinition gelten soll.
- `div > p { color:blue; }`  
Mit dem Zeichen `>` wird in diesem Selektor festgelegt, dass `p`-Elemente, sofern sie innerhalb eines `div`-Bereichs vorkommen, blaue Textfarbe (`color:blue;`) erhalten. Aber nur dann, wenn das `p`-Element in dieser Form vorkommt: `<div><p></p></div>`, also eine Ebene unterhalb des `div`-Elements. Im obigen Beispiel ist innerhalb des `div`-Bereichs unter anderem auch eine Tabelle notiert, in der noch mal ein `p`-Element vorkommt. Dort greift die Formatierung der blauen Farbe nicht, weil das `p`-Element mehr als eine Verschachtelungsebene unterhalb des `div`-Elements liegt (die Tabellenelemente sind dazwischen).
- `div * b { color:violet; }`  
Mit dem Zeichen `*` wird in diesem Selektor festgelegt, dass `b`-Elemente, sofern sie innerhalb eines `div`-Bereichs vorkommen, violette Textfarbe (`color:violet;`) erhalten. Aber nur dann, wenn das `b`-Element in dieser oder einer vergleichbaren Form vorkommt: `<div><p><b></b></p></div>`, also mindestens zwei Ebenen unterhalb des `div`-Elements. Im obigen Beispiel sind verschiedene Textstellen an verschiedenen Stellen mit `<b>...</b>` ausgezeichnet. Das erste `b`-Element in dem Text, der zuerst und direkt innerhalb des `div`-Bereichs notiert ist, erhält noch keine violette Farbe, wohl aber das nächste `b`-Element im gleichen Satz, da dieses Element bereits zwei Ebenen unterhalb des `div`-Elements liegt (das `i`-Element liegt dazwischen).
- `div + p { margin-top:100px; }`  
Mit dem Zeichen `+` wird in diesem Selektor festgelegt, dass `p`-Elemente, die unmittelbar auf ein `div`-Element hin folgen, einen Abstand von 100 Pixeln nach oben hin erhalten (`margin-top:100px;`). Im obigen Beispiel ist das bei dem vorletzten `p`-Element der Fall. Dieses folgt unmittelbar hinter dem `div`-Bereich. Beachten Sie, dass die Syntax mit dem Pluszeichen nur mit Netscape 6.0 nachvollziehbar war, mit dem Internet Explorer 5.5 dagegen nicht.

### 15.3.4 Attributbedingte Formate definieren

Mit Hilfe attributbedingter Formate können Sie in Selektoren angeben, dass Formatdefinitionen nur für Elemente mit bestimmten Attributen oder sogar nur für Elemente mit bestimmten Wertzuweisungen an Attribute gelten sollen. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
p { font-weight:bold; font-family:Tahoma,sans-serif; font-size:14pt; }
p[align] { color:red; }
```

```

p[align=center] { color:blue; text-align:left; }
td[abbr~=Berlin] { background-color:#FFFF00 }
*[lang|=en] { background-color:#FF0000; color:#FFFFFF; }
-->
</style>
</head>
<body>
<p>Das ist ein Textabsatz, ein stinknormaler.</p>
<p align="right">Das ist ein Textabsatz, rechts ausgerichtet.</p>
<p align="center">Das ist ein Textabsatz, zentriert? Wirklich zentriert?</p>
<table border="1"><tr>
  <th>Berlin</th>
  <th>Hamburg</th>
</tr><tr>
  <td abbr="Es folgen Infos zu Berlin">Eine Menge Inhalt zu Berlin</td>
  <td abbr="Es folgen Infos zu Hamburg">Eine Menge Inhalt zu Hamburg</td>
</tr></table>
<p>Ein Textabsatz <span lang="fr">avec</span> Text <span lang="en">about</span>
einen <span lang="en-US">English man in New York</span></p>
</body>
</html>

```

**Das ist ein Textabsatz, ein stinknormaler.**

**Das ist ein Textabsatz, rechts ausgerichtet.**

**Das ist ein Textabsatz, zentriert? Wirklich zentriert?**

| Berlin                      | Hamburg                      |
|-----------------------------|------------------------------|
| Eine Menge Inhalt zu Berlin | Eine Menge Inhalt zu Hamburg |

**Ein Textabsatz avec Text **about** einen **English man in New York****

15.3: Attributbedingte Formate unterstützt Netscape 6, nicht aber Internet Explorer (zumindest bis zur Version 6).

In dem Beispiel ist zunächst eine Formatdefinition für alle p-Elemente notiert. Dann folgen zwei weitere Formatdefinitionen – ebenfalls für p-Elemente, aber nur für solche, die ein Attribut `align` haben. Mit der Syntax `p[align]` werden alle p-Elemente erfasst, die ein `align`-Attribut haben. Mit `p[align=center]` werden alle p-Elemente erfasst, die ein `align="center"` im einleitenden Tag haben. Mit `td[abbr~=Berlin]` werden alle Tabellenzellen des Typs `td` erfasst, die ein Attribut `abbr` besitzen und bei denen in der Wertzuweisung an dieses Attribut das Wort Berlin vorkommt. Diese Syntax findet eines von mehreren, durch Leerzeichen getrennten Wörtern in Wertzuweisungen an Attribute. Mit `*[lang|=en]` schließlich werden alle Elemente (der Stern ist das Platzhalterzeichen für »alle Elemente«) erfasst, die ein Universalattribut `lang`=

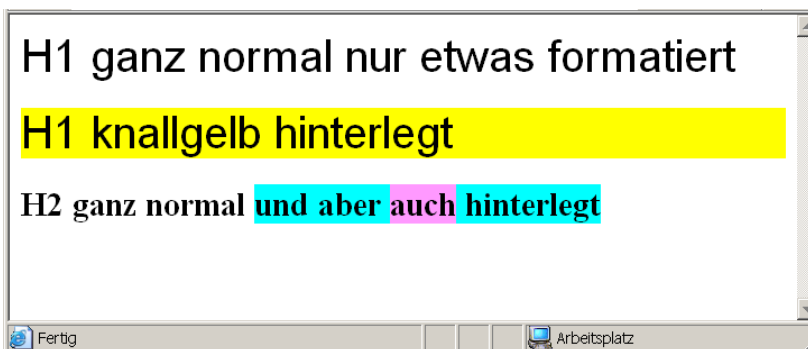
mit einem Wert wie `en` oder auch `en-US` haben. Diese Syntax findet Teile in Wertzuweisungen an Attribute, die durch Bindestriche getrennt sind.

**Beachten Sie:** Netscape 6.0 interpretiert diese Angaben, der Internet Explorer 5.5 und 6.0 noch nicht.

### 15.3.5 Formate für Klassen definieren

Sie können Formate für Klassen definieren. Anwenden können Sie solche Klassen in HTML mit dem Universalattribut `class`. So können Sie beispielsweise für Überschriften 1. Ordnung zwei Varianten erzeugen. Dazu vergeben Sie Namen für die Klassen. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
h1 { font-family:Arial,sans-serif; font-size:24pt; font-weight:normal; }
h1.hinterlegt { background-color:#FFFF00 }
*.hinterlegt { background-color:#00FFFF }
.extra { background-color:#FF99FF }
-->
</style>
</head>
<body>
<h1>H1 ganz normal nur etwas formatiert</h1>
<h1 class="hinterlegt">H1 knallgelb hinterlegt</h1>
<h2>H2 ganz normal <span class="hinterlegt">und aber <b class="extra">auch</b>
hinterlegt</span></h2>
</body>
</html>
```



15.4: Die CSS-Formate werden nur bei HTML-Elementen mit einem bestimmten Attributwert bei `class` wirksam.

Es gibt zwei Möglichkeiten, Klassen für HTML-Elemente zu notieren: entweder für einen bestimmten HTML-Elementtyp oder für keinen bestimmten. In jedem Fall bestimmen Sie eine Formatdefinition für eine Klasse, indem Sie im Selektor einen Punkt notieren und unmittelbar dahinter einen Namen für die Klasse angeben. Im obigen Beispiel wird mit `h1.hinterlegt` eine Klasse namens `hinterlegt` angesprochen, die aber nur für HTML-Elemente vom Typ `h1` gilt, also z.B. auf HTML-Elemente wie `<h1 class="hinterlegt">` anwendbar ist. Mit `*.hinterlegt` wird im Beispiel demonstriert, dass Sie durchaus den gleichen Klassennamen noch mal allgemein oder für andere Elemente verwenden können. Der Stern gilt als Platzhalterzeichen und bedeutet so viel wie »für alle Elemente«. Er kann auch ganz entfallen. Im Beispiel `.extra` können Sie das sehen. Ein `h1`-Element erhält im Beispiel also die Formatierungen, die mit dem Selektor `h1.hinterlegt` bestimmt werden. Ein `p`-Element mit `<p class="hinterlegt">` erhält dagegen die Formatierungen, die mit `*.hinterlegt` bestimmt werden.

Die Namen hinter dem Punkt können Sie frei vergeben. Die Namen sollten nicht zu lang sein und keine Leerzeichen und keine deutschen Umlaute enthalten.

### 15.3.6 Klassen in Verbindung mit den Elementen `div` und `span`

Die beiden HTML-Elemente `div` und `span` haben besondere Bedeutung für CSS Stylesheets. Der Grund ist, dass sie selber relativ eigenschaftslos sind. Der einzige Unterschied besteht darin, dass das `div`-Element eine neue Zeile im Textfluss erzwingt, während `span` zur Verwendung innerhalb eines Textes verwendet werden kann und daher überhaupt keine Eigenschaften hat. Wenn Sie mit diesen Elementen arbeiten, können Sie Probleme vermeiden, die sich aus den im Browser voreingestellten Eigenschaften bestimmter Elemente ergeben. So haben beispielsweise Überschriften bei der Darstellung im Browser bestimmte Vor- und Nachabstände, die von Browser zu Browser recht verschieden sein können. Wenn Sie das Aussehen solcher Elemente kontrollieren wollen, benötigen Sie CSS-Formatdefinitionen nicht selten dazu, um einfach nur die automatischen Voreinstellungen der Browser für solche Elemente zu korrigieren. Bei den Elementen `div` und `span` können Sie dagegen fast wie auf einer »tabula rasa« aufsetzen.

Das folgende Beispiel zeigt, wie Sie mit CSS, Klassen und allein mit diesen beiden eigenschaftslosen HTML-Elementen arbeiten können:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
.body { background-color:#EEEEFF; }
.chap { font-family:Arial,sans-serif; font-size:20pt; color:blue;
        border-bottom-style:solid; border-bottom-width:3px; border-color:red;
        margin:0px; margin-bottom:16px; }
.sect { font-family:Arial,sans-serif; font-size:16pt; color:blue;
        border-bottom-style:solid; border-bottom-width:3px; border-color:red;
        margin:0px; margin-top:24px; margin-bottom:16px; }
```

```

.p { font-family:Arial,sans-serif; font-size:11pt; color:black; margin-top:6px;
margin-bottom:6px; }
.red { color:red }
.blue { color:blue }
.bold { font-weight:bold }
.big { font-size:14pt }
.small { font-size:6pt; font-family:'Small Fonts',sans-serif; }
-->
</style>
</head>
<body>
<div class="chap">Eine Kapitelüberschrift</div>
<div class="p">Eine Textabsatz, dessen Klasse genau so heißt wie ein HTML-Element
:-)</div>
<div class="p">Und damit man auch alles sieht, gleich noch einer davon</div>
<div class="sect">Eine Sektionsüberschrift</div>
<div class="p">Absatz mit <span class="red">rotem und <span
class="bold">fettem</span></span> Text</div>
<div class="p">Absatz mit <span class="big">großem</span> und <span
class="small">kleinem</span> Text</div>
</body>
</html>

```



15.5: Mit `div` und `span` formatieren Sie Bereiche mit bestimmten CSS-Eigenschaften – das Beispiel enthält außer diesen beiden keinerlei Tags zum Formatieren.

Im `style`-Bereich des Beispiels werden Klassen für beliebige Elemente definiert, die diese Klassen mit `class="chap"`, `class="sect"` usw. verwenden können und damit die entsprechenden Formate annehmen.

Im sichtbaren Bereich der Datei, also zwischen `<body>` und `</body>`, sind ausschließlich Elemente der Typen `div` und `span` notiert. Durch die Verwendung des `class`-Attributs »simulieren« diese Elemente gewissermaßen typische Auszeichnungselemente wie Kapitelüberschriften, Textabsätze und Formatierungen innerhalb des Textes.



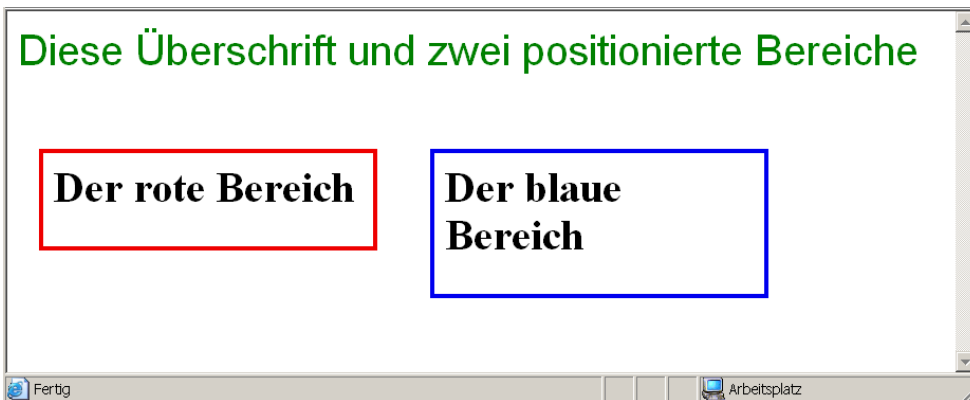
Nun kann man darüber streiten, ob diese Art zu arbeiten nicht eine Vergewaltigung der Idee von HTML darstellt, der Idee nämlich, ein Set wichtiger und für normalen Text typischer Auszeichnungselemente bereitzustellen, damit durch deren einheitliche Verwendung auch eine »webweit« einheitliche Semantik bei der Textauszeichnung erreicht wird. Andererseits ist Semantik immer auch gegeben, wenn man Elemente regelgerecht, aber nach eigenen Vorstellungen für Bedeutung auszeichnet. Denn vielleicht will man ja unter der Bedeutung der Klassen `chap` und `sect` im Beispiel etwas anderes verstehen als einfach unter dem, was die HTML-Elemente `h1` und `h2` bedeuten. Die eigenschaftslosen Elemente `div` und `span` eignen sich ja beispielsweise auch hervorragend dazu, in Verbindung mit Klassennamen wie `Romeo` und `Julia` Rollentexte eines Theaterstücks zu markieren. Insofern sind `div` und `span` in Verbindung mit Klassennamen gar nicht mehr weit entfernt von den semantischen Konzepten, die mit XML verfolgt werden.

### 15.3.7 Individualformate definieren

So wie Sie Formate für Klassen definieren können, die in HTML mit dem Universalattribut `class=` angesprochen werden, können Sie auch Formate definieren, die über das Universalattribut `id=` angesprochen werden. Da die Wertzuweisung an ein solches Attribut ein dokumentweit eindeutiger Name sein sollte, handelt es sich also einfach um eine zentrale Formatdefinition für das eine Element mit diesem `id`-Namen. In CSS werden die `id`-Namen allerdings nicht nur wie beispielsweise in JavaScript als dokumentweit eindeutige Bezeichner betrachtet, sondern auch als elementtypweit eindeutige Bezeichner. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
#roterBereich {
    position:absolute;
    top:130px;
    left:30px;
    width:320px;
    padding:10px;
    margin:0px;
    border:4px solid #EE0000;
}
#blauerBereich {
    position:absolute;
    top:130px;
    left:400px;
    width:320px;
    padding:10px;
    margin:0px;
    border:4px solid #0000EE;
}
h1#Titel {
    font-family:Arial,sans-serif;
```

```
font-size:24pt;
font-weight:normal;
color:green;
}
-->
</style>
</head>
<body>
<h1 id="Titel">Diese Überschrift und zwei positionierte Bereiche</h1>
<div id="roterBereich"><h1>Der rote Bereich</h1></div>
<div id="blauerBereich"><h1>Der blaue Bereich</h1></div>
</body>
</html>
```



15.6: Individualformate gelten für die HTML-Elemente mit dem passenden Attributwert bei id.

In dem Beispiel werden drei Individualformate definiert. Zunächst einmal sind dies zwei Individualformate mit den Namen `#roterBereich` und `#blauerBereich`. Solche Formate beginnen also mit dem Gatterzeichen `#`, gefolgt von dem Namen. Ein HTML-Element, das diesen Namen als Wertzuweisung an das `id`-Attribut benutzt, bekommt dann die entsprechenden Formate zugewiesen. Das obige Beispiel zeigt einen typischen Anwendungsfall für Individualformate, nämlich das Definieren von Formaten für absolut positionierte Bereiche (`position:absolute`).

Außerdem wird ein Individualformat mit dem Selektor `h1#Titel` definiert. Das ist eine Syntax, die erst seit CSS 2.0 zur Verfügung steht. Dabei wird vor dem Gatterzeichen, das ein Individualformat kennzeichnet, ein HTML-Elementtyp notiert, im Beispiel also `h1` für Überschriften erster Ordnung. Hinter dem Gatterzeichen steht dann wieder der Name des Individualformats. Im Beispiel wird die Formatdefinition nur dann angewendet, wenn es in der HTML-Datei ein `h1`-Element mit dem Attribut `id="Titel"` gibt. Wenn es dagegen etwa ein Element `div` mit `id="Titel"` gäbe, würde die Formatdefinition dort ignoriert.

**Beachten Sie:** Individualformate mit vorangestelltem Elementtyp werden von Internet Explorer 5.x und Netscape 6.0 erkannt, nicht aber von älteren Versionen.

### 15.3.8 Pseudoformate definieren

Pseudoformate können Sie für HTML-Bestandteile definieren, die sich jedoch nicht durch ein eindeutiges HTML-Element ausdrücken lassen, z.B. ein »noch nicht besuchter Verweis« oder der »erste Buchstabe eines Absatzes«. Zum Definieren solcher Pseudoformate gibt es eine spezielle Syntax. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
a:link { color:#EE0000; text-decoration:none; font-weight:bold; }
a:visited { color:#EEAAAA; text-decoration:none; font-weight:bold; }
a:hover { color:#EE0000; text-decoration:none; background-color:#FFFF99; font-
weight:bold; }
a:active { color:#0000EE; background-color:#FFFF99; font-weight:bold; }
body { font-family:Arial,sans-serif; font-size:12pt; }
-->
</style>
</head>
<body>
<p>
Vermutlich kennen Sie die <a href="../../index.htm">Kapiteleinstiegsseite zu
CSS</a><br>
Und wohl auch die <a href="../../index.htm">Einstiegsseite zu SELFHTML</a><br>
Aber kennen Sie auch die <a
href="../../intro/hypertext/geschichte.htm">Geschichte von Hypertext</a>?<br>
Oder die <a href="../../xml/regeln/baumstruktur.htm">Baumstruktur einer XML-
Datei</a>?
</p>
</body>
</html>
```



15.7: Pseudoformate kommen vor allem bei Links zum Einsatz.

Bei Pseudoformaten notieren Sie zuerst das betreffende HTML-Element, im Beispiel das a-Element für Verweise. Dahinter folgt ein Doppelpunkt und dahinter eine erlaubte Angabe, im Beispiel etwa `link` (für noch nicht besuchte Verweisziele), `visited` (für bereits besuchte Verweisziele), `hover` (für Verweise, während der Anwender mit der Maus darüber fährt) und `active`

(für angeklickte Verweise). Beachten Sie, dass dies keine frei wählbaren Namen sind, sondern feste Schlüsselwörter.

Es gibt eine Reihe fester, erlaubter Pseudoformate. Diese werden im Zusammenhang mit den CSS-Eigenschaften im Abschnitt 16.10 beschrieben.

## 15.4 HTML-Elemente direktformatieren

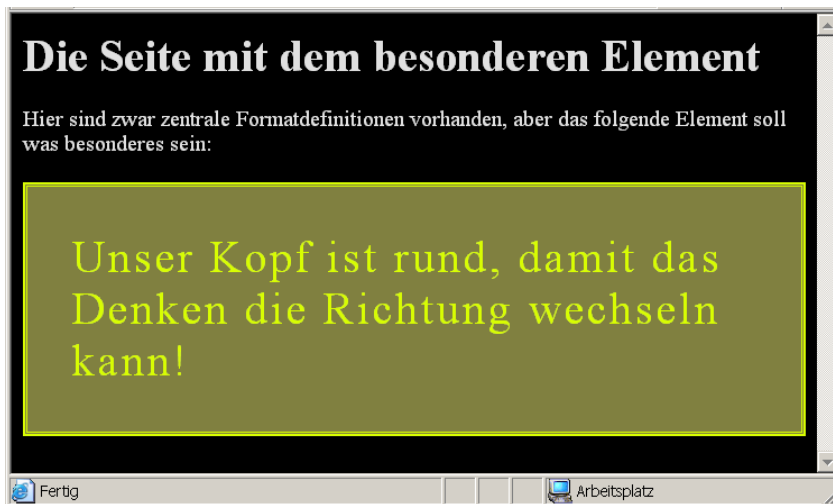
### 15.4.1 Formatdefinitionen mit dem style-Attribut

Sie können einzelne HTML-Elemente innerhalb des sichtbaren HTML-Dokuments mit Hilfe von Stylesheet-Eigenschaften formatieren. Die Formatdefinitionen gelten dann genau für den Geltungsbereich des betreffenden HTML-Elements und nach den Regeln der natürlichen Formatvererbung auch für Kindelemente des betreffenden Elements. Sinnvoll ist diese Möglichkeit, wenn Sie entweder sonst auf CSS verzichten und es nur mal für ein paar Ausnahmen benötigen, oder wenn Sie zentrale Formate verwenden, einzelne Elemente aber anders als mit den dort definierten Formatdefinitionen gestalten möchten. Ein Beispiel:

```
<html>
<head>
<title>Titel der Datei</title>
<style type="text/css">
<!--
body { background-color:#000000; color:#E0E0E0 }
-->
</style>
</head>
<body>
<h1>Die Seite mit dem besonderen Element</h1>
<p>Hier sind zwar zentrale Formatdefinitionen vorhanden,
aber das folgende Element soll was besonderes sein:</p>

<p style="background-color:#808040; color:#D8FD02;
font-family:'Century Schoolbook',serif; font-size:24pt; letter-spacing:3px;
padding:40px; border:double #D8FD02 4px;"
title="Zitat von Francis Picabia">
Unser Kopf ist rund, damit das Denken die Richtung wechseln kann!
</p>

</body>
</html>
```



15.8: Mit dem Attribut `style` formatieren Sie HTML-Tags individuell.

Sie können Formate für ein HTML-Element definieren, indem Sie innerhalb des einleitenden HTML-Tags das Attribut `style=` und dahinter die gewünschten Formatdefinitionen notieren. Innerhalb der Formatdefinitionen sind CSS-Eigenschaften erlaubt. Dabei gilt die gleiche Syntax wie innerhalb der geschweiften Klammern bei zentralen Formaten. Jede Eigenschaft besteht aus einem Namen, z.B. `color` (Schriftfarbe), und einem Wert, z.B. `#D8FD02`, getrennt durch einen Doppelpunkt. Schließen Sie jede Formatdefinition jeweils durch einen Strichpunkt ab. Nur bei der letzten Formatdefinition vor dem abschließenden Anführungszeichen darf der Strichpunkt entfallen.

Im obigen Beispiel werden die individuellen Formatangaben auf ein gewöhnliches `p`-Element angewendet. Es erhält mit Hilfe der CSS-Eigenschaften für Hintergrundfarbe (`background-color`), Schriftfarbe (`color`), Schriftart (`font-family`), Schriftgröße (`font-size`), Zeichenabstand (`letter-spacing`), Innenabstand zum Rand (`padding`) und einem Rahmen (`border`) ein Aussehen, das überhaupt nicht mehr an einen »Textabsatz« erinnert.

## 15.5 Maßeinheiten, Farbangaben und Wertzuweisung

### 15.5.1 Numerische Angaben

Bei numerischen Angaben zu CSS-Eigenschaften, also etwa bei Schriftgrößen, Absatzabständen oder Rändern, stehen Ihnen die weit verbreiteten Maßeinheiten zur Verfügung. Dabei sind absolute Angaben (z.B. Millimeter) und relative Angaben (z.B. Prozent gegenüber »normal«) möglich.

Benutzen Sie für Ihre Angaben die folgenden Abkürzungen:

| Abkürzung | Angabetyp       | Bedeutung                                                                                                                                                                                                                                                                         | Beispiele                                     |
|-----------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| pt        | absolut         | Steht für Punkt. Typographische Maßeinheit. 1 Punkt entspricht 1/72 Inches.                                                                                                                                                                                                       | font-size:12pt;<br>line-height:14pt;          |
| pc        | absolut         | Steht für Pica. Typographische Maßeinheit. 1 Pica entspricht 12 Punkt.                                                                                                                                                                                                            | font-size:1pc;<br>line-height:1.2pc;          |
| in        | absolut         | Steht für Inch. Allgemeines Längenmaß im angelsächsischen Raum. 1 Inch entspricht 2.54 Zentimetern.                                                                                                                                                                               | border-width:0.1in;<br>margin-left:1in;       |
| mm        | absolut         | Steht für Millimeter. Allgemeines Längenmaß. 1000. Teil des Urmeters.                                                                                                                                                                                                             | margin-bottom:10mm;<br>width:70mm;            |
| cm        | absolut         | Steht für Zentimeter. Allgemeines Längenmaß. 100. Teil des Urmeters.                                                                                                                                                                                                              | font-size:1cm;<br>top:2.54cm;                 |
| px        | absolut/relativ | Steht für Pixel. Abhängig von der Pixeldichte des Ausgabegeräts, relativ also von Ausgabegerät zu Ausgabegerät, absolut dagegen auf ein und das selbe Ausgabegerät bezogen.                                                                                                       | border-width:3px;<br>margin-right:60px;       |
| em        | relativ         | Steht für bezogen auf die Schriftgröße des Elements. Ausnahme: Wenn die font-size-Eigenschaft (also die Schriftgröße) selbst mit dieser Maßangabe gesetzt wird, steht diese für bezogen auf die Schriftgröße des Elternelements.                                                  | font-size:1.2em;<br>line-height:1.5em;        |
| ex        | relativ         | Steht für bezogen auf die Höhe des Kleinbuchstaben x in diesem Element. Ausnahme: Wenn die font-size-Eigenschaft (also die Schriftgröße) selbst mit dieser Maßangabe gesetzt wird, steht diese für bezogen auf die Höhe des Kleinbuchstaben x im Elternelement.                   | text-transform:uppercase;<br>font-size:1.3ex; |
| %         | relativ         | Steht für Prozent. Je nach CSS-Eigenschaft relativ zur elementeigenen Größe, oder zu der des Elternelements, oder zu einem allgemeineren Kontext. Bei nebenstehendem Beispiel ist die Angabe zur Zeilenhöhe (line-height) relativ zur Schriftgröße (font-size) zu interpretieren. | font-size:10pt;<br>line-height:120%;          |

Benutzen Sie bei numerischen Bruchzahlen stets den Punkt als Dezimalzeichen, nicht das deutsche Komma, also etwa 0.2cm und nicht 0,2cm. Auch Minuswerte sind bei einigen Wertzuweisungen möglich, markiert wie üblich durch das Minuszeichen.

Sie können also sehr exakte Angaben notieren, doch Web-Seiten werden in der Regel auf Bildschirmen ausgegeben, und Bildschirme bestehen heutzutage aus Pixeln, wobei die Pixeldichte der einzelnen Bildschirme sehr unterschiedlich sein kann. Der Computer muss Ihre Angaben für

die Ausgabe am Bildschirm letztlich in Pixel umrechnen. Dabei kann es zu enttäuschenden Ergebnissen kommen. So erscheinen Schriftgrößen, die am einen Bildschirm gut lesbar sind, in einer anderen Umgebung zu klein. Leider sind daran auch die unterschiedlichen Basiseinstellungen der Betriebssysteme mit schuld. So wird unter Windows-Systemen eine Berechnungsgrundlage von 96 dpi (*Dots per Inch*) für die Darstellung von Schriftarten verwendet (bei der Einstellung »große Schriftarten« gar 120 dpi), unter Systemen wie Linux und Macintosh sind es dagegen 72 dpi. Deshalb erscheinen Schriftgrößen, die unter Windows sauber aussehen, unter den anderen genannten Systemen oft mickrig, während Schriftgrößen, die dort angenehm aussehen, unter Windows wiederum klobig wirken. Aber auch direkte Angaben in Pixeln lösen das Problem nicht unbedingt. Ein Pixel auf einem 13-Zoll-Notebook-Bildschirm mit 1024×768 Pixeln Bildschirmauflösung wirkt ganz anders als ein Pixel auf einem 19-Zoll-Desktop-Bildschirm mit der gleichen Bildschirmauflösung. Andere Angaben wie `em` oder Prozentwerte umgehen zwar das Problem der unterschiedlichen Bildschirmdarstellungen – aber erstens kann man bei einer Angabe wie `1.2em` kaum von »Formatierung« reden, und zweitens bereiten solche Angaben auch andere Probleme. So kann es beispielsweise durch das Prinzip der natürlichen Vererbung passieren, dass Schriftarten bei mehrfach verschachtelten Elementen (etwa bei Listen oder Tabellen) immer kleiner oder größer werden. CSS 2.0 bietet zwar mit den erweiterten Möglichkeiten, Formate für verschachtelte Elemente zu definieren, Maßnahmen zur Abhilfe an, doch das nutzt vorläufig auch nichts, solange es in vielen noch verbreiteten Browsern nicht funktioniert.

Um die »richtigen« und »falschen« Maßangaben ist angesichts all dieser Probleme in einigen Kreisen schon ein richtiger Glaubenskrieg ausgebrochen. Die einen behaupten, man solle sich an Angaben wie `pt` halten, weil ein Punkt immer gleich groß zu sein hat, die anderen beschwören die `em`-Angabe wie einen Erlösergott. Fakt ist in jedem Fall, dass Sie niemals darauf vertrauen sollten, dass Größen, die auf Ihrem Bildschirm wunderbar aussehen, woanders genauso wunderbar wirken. Vermeiden Sie deshalb vor allem bei Angaben, die für die Leserlichkeit von Text relevant sind, »extreme« Werte, und reagieren Sie, wenn Sie von vielen Seiten zu hören bekommen, Ihre Seiten seien schwer lesbar.

### 15.5.2 Farbangaben

Farbangaben in Stylesheets nach CSS-Syntax sind nach den Regeln zum Definieren von Farben in HTML (Kapitel 3.5) möglich. Sie können Farbwerte also hexadezimal nach dem Schema `#XXXXXX` angeben, aber auch die verbreiteten Farbnamen verwenden.

Eine weitere Möglichkeit, Farben zu definieren, stellt die CSS-Stylesheet-Sprache jedoch auch noch zur Verfügung. Das Schema dazu lautet `rgb(rrr,ggg,bbb)`. Der Ausdruck `rgb(...)` ist dabei immer gleich. Innerhalb der Klammern müssen drei Dezimalwerte oder drei Prozentwerte stehen, durch Kommata voneinander getrennt. `rrr` ist der Rotwert, `ggg` der Grünwert und `bbb` der Blauwert der Farbe. Für alle drei Werte sind entweder absolute Zahlen zwischen 0 (kein Anteil der entsprechenden Farbe) und 255 (maximaler Anteil der entsprechenden Farbe) oder Prozentwerte von 0% (kein Anteil an der entsprechenden Farbe) bis 100% (maximaler Anteil der entsprechenden Farbe) erlaubt. Einige Beispiele:

```
<style type="text/css">
<!--
h1 { color:#FFCCDD; background-color:black }
body { background-color:rgb(51,0,102); }
```

```
blockquote { background-color:rgb(60%,90%,75%); }
//-->
</style>
```



### Anwenderrelative Farbwörter

Ab der CSS-Version 2.0 gibt es ferner die Möglichkeit, anwenderrelative Farbwörter zu verwenden, hinter denen sich Farben verbergen, die der Anwender an seinem Rechner als Arbeitsumgebung eingestellt hat. Auf diese Weise können Sie zur Gestaltung Ihrer Inhalte Farben verwenden, die auf die Bildschirmumgebungsfarben des Anwenderrechners abgestimmt sind, ohne dass Sie diese Farben kennen. Die erlaubten anwenderrelativen Farbwörter werden unterhalb des folgenden Beispiels erläutert.

| Abkürzung         | Bedeutung                                                       |
|-------------------|-----------------------------------------------------------------|
| activeborder      | Farbe der aktiven Fenstertitelzeile.                            |
| activeborder      | Farbe der aktiven Fenstertitelzeile.                            |
| activecaption     | Farbe der Überschrift in der aktiven Fenstertitelzeile.         |
| appworkspace      | Farbe des Hintergrunds der aktiven Anwendung.                   |
| background        | Farbe des Desktop-Hintergrunds.                                 |
| buttonface        | Farbe von Buttons in Dialogfenstern.                            |
| buttonhighlight   | Farbe für 3-D-Schatten von Buttons in Dialogfenstern.           |
| buttontext        | Farbe von Texten beschrifteter Buttons in Dialogfenstern.       |
| captiontext       | Farbe von Überschriften in Dialogfenstern.                      |
| greytext          | Farbe von deaktiviertem Text in Dialogfenstern.                 |
| highlight         | Farbe von ausgewählten Einträgen in Auswahllisten.              |
| highlighttext     | Farbe von selektiertem Text.                                    |
| inactiveborder    | Farbe einer nicht aktiven Fenstertitelzeile.                    |
| inactivecaption   | Farbe der Überschrift in einer nicht aktiven Fenstertitelzeile. |
| infobackground    | Farbe für Tooltips und Hints (kleine Popup-Hilfen).             |
| infotext          | Textfarbe für Tooltips und Hints (kleine Popup-Hilfen).         |
| menu              | Farbe für Menüleisten.                                          |
| menutext          | Farbe für Menüeinträge.                                         |
| scrollbar         | Farbe der Scroll-Leiste in Fenstern.                            |
| threeddarkshadow  | Dunkle Farbe bei Schatten von 3-D-Elementen.                    |
| threedface        | Farbe von 3-D-Elementen.                                        |
| threedhighlight   | Farbe von gerade angeklickten 3-D-Elementen.                    |
| threedlightshadow | Helle Farbe bei Schatten von 3-D-Elementen.                     |
| threedshadow      | Dunkle Farbe bei Schatten von 3-D-Elementen.                    |
| window            | Hintergrundfarbe von Dokumentfenstern.                          |
| windowframe       | Farbe von Fensterrahmen.                                        |
| windowtext        | Farbe von normalem Text in Dokumentfenstern.                    |



Einige Beispiele:

```
<style type="text/css">
<!--
h1 { color:windowtext; background-color:infobackground; }
body { background-color:appworkspace; }
/-->
</style>
```

### 15.5.3 Erlaubte feste Werte und Zeichenkettenangaben

Bei vielen CSS-Eigenschaften steht eine Auswahl erlaubter Werte zur Verfügung. Das ist nicht anders als bei vielen HTML-Attributen. Genau so, wie Sie etwa eine Überschrift oder einen Textabsatz in HTML mit den Angaben `align="left"`, `align="center"` oder `align="right"` ausrichten können, gibt es CSS-Eigenschaften, die bestimmte Schlüsselwörter erwarten. So gibt es etwa die Stylesheet-Angabe `text-align` zur Ausrichtung von Text, die die gleichen Angaben erwartet, also `left`, `center` oder `right`.

Manche Stylesheet-Angaben erwarten die Angabe eines Namens oder auch mehrerer Schlüsselwörter. So können Sie hinter der Stylesheet-Angabe `font-family` Namen von einer oder mehreren Schriftarten notieren, die Sie für die Formatierung verwenden möchten. Andere CSS-Eigenschaften sind Zusammenfassungen von Einzeleigenschaften. So gibt es beispielsweise die Angabe `border` (Rahmen), hinter der Sie eine Angabe wie `thin solid red` notieren können. Trennen Sie solche Angaben zu einer Sammeleigenschaft durch Leerzeichen. Wenn die Angaben selbst Leerzeichen enthalten können, wie etwa bei Namen von Schriftarten, können Sie Kommata zum Trennen verwenden. In solchen Fällen sind auch Anführungszeichen empfehlenswert.

Bei Namen wie Schriftartangaben oder Angaben wie `left`, `center` usw. ist es zumindest bei der CSS-Anwendung auf HTML-Elemente egal, ob Sie die Angaben groß oder klein schreiben. Das gilt auch für die Stylesheet-Eigenschaften selbst. Wenn Sie CSS allerdings auf XHTML oder gültige XML-Dokumente anwenden, müssen Elementnamen, Attributnamen und Attributzuweisungen so geschrieben werden, wie es in der XML-DTD festgelegt ist. Bei XHTML muss alles klein geschrieben werden.

Einige Beispiele:

```
<style type="text/css">
<!--
p { font-family:"Comic Sans MS",Arial; text-align:center; }
body { border:thin dashed silver; background-color:black; }
DIV.SPEZIAL { BORDER:THICK DOUBLE RGB(0,64,128); }
blockquote[title="Zitat von Goethe"] { color:#000080; }
/-->
</style>
```

### 15.5.4 Numerische Angaben für Sprachausgabe

Für CSS-Eigenschaften der Sound-Kontrolle für Sprachausgabe, die von den Browsern jedoch bislang nicht unterstützt wird, werden die folgenden Maßangaben aus dem Bereich der Akustik und Zeitmessung verwendet:

| Abkürzung | Bedeutung                                                                                 |
|-----------|-------------------------------------------------------------------------------------------|
| deg       | Kreisgrad, wobei der Kreis in 360 Grad aufgeteilt ist. Ein rechter Winkel hat z.B. 90deg. |
| grad      | Gradient, Maßangabe für Steigungen und Gefälle.                                           |
| rad       | Radian, geometrische Maßangabe.                                                           |
| ms        | Millisekunden, Maßangabe für Zeiteinheit.                                                 |
| s         | Sekunden, Maßangabe für Zeiteinheit.                                                      |
| Hz        | Hertz, Maßangabe für Frequenzen.                                                          |
| kHz       | kiloHertz, Maßangabe für Frequenzen.                                                      |

## 16 CSS-Eigenschaften

### 16.1 Schriftformatierung

#### *Allgemeines zur Schriftformatierung*

Unter Schriftformatierung sind Angaben zu Schriftarten, Schriftgrößen, Schriftfarben, Schriftgewicht, Zeichen- und Wortabständen usw. zu verstehen. Sinnvoll sind solche Angaben für alle HTML-Elemente, die Text enthalten können. Dazu gehören alle Elemente, die im Kapitel 5 *Text strukturieren* beschrieben werden, aber auch beispielsweise Elemente für Tabellen. Auch auf das `body`-Element lassen sich die hier aufgelisteten Stylesheet-Angaben anwenden – in diesem Fall gelten die Angaben für alle Textelemente der HTML-Datei.



#### *font-family (Schriftart)*

Unter Schriftarten sind Schriftarten wie etwa Arial, Helvetica, Times Roman usw. zu verstehen. Auch Schriftfamilien wie Sans Serif usw. gehören dazu.

Bei der hier beschriebenen Eigenschaft können Sie Schriftarten angeben, ohne sich darum zu kümmern, ob und wie die Schriftart beim Anwender angezeigt werden kann. Falls die angegebene Schriftart nicht angezeigt werden kann, bleibt die Angabe wirkungslos. Eine Möglichkeit, Schriftarten durch Angabe einer bestimmten Schriftarten-Datenquelle zu erzwingen, wird im Abschnitt 16.2 *Schriftformatierung mit Schriftartendatei* beschrieben. Ein Beispiel:

```
<html><head><title>font-family</title>
<body bgcolor="#FFFFFF" text="#000000">

<p><span style="font-family:'Times New Roman',Times,serif">
kleiner Beispieltext in Times New Roman</span><br>
<span style="font-family:'Times New Roman',Times,serif;font-size:200%">
großer Beispieltext in Times New Roman</span></p>

<p><span style="font-family:Verdana">kleiner Beispieltext
in Verdana</span><br>
<span style="font-family:Verdana;font-size:200%">großer Beispieltext in
Verdana</span></p>

<!--usw.-->

</body></html>
```



16.1: Das Bild zeigt eine Reihe von Schriften, die unter Windows standardmäßig installiert sind.

Mit `font-family`: können Sie eine oder mehrere Schriftarten bestimmen. Bei mehreren angegebenen Schriftarten ist die Reihenfolge der Angabe entscheidend: Ist die erste angegebene Schriftart verfügbar, wird diese verwendet. Ist sie nicht verfügbar, wird die zweite Schriftart verwendet, falls diese verfügbar ist, usw.

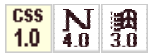
Trennen Sie mehrere Schriftartennamen durch Kommata. Das W3-Konsortium empfiehlt, Schriftartennamen, die Leerzeichen enthalten, in Anführungszeichen zu setzen, also z.B. `font-family: "Century Schoolbook", Times`. Sie können dazu die doppelten oder die einfachen Anführungszeichen (Hochkomma) verwenden. Falls die CSS-Eigenschaften wie im obigen Beispiel innerhalb eines `style`-Attributs im HTML-Quelltext vorkommen, dürfen Sie nicht die gleichen Anführungszeichen verwenden wie bei der HTML-Attributzuweisung. Im obigen Beispiel sehen Sie, dass die Wertzuweisung an das `style`-Attribut wie in doppelten Anführungszeichen steht. Innerhalb davon werden für 'Times New Roman' deshalb einfache Anführungszeichen verwendet, um Konflikte zu vermeiden.

Folgende **generische** Schriftfamilien sind fest vordefiniert – diese Angaben können Sie also neben Schriftartennamen benutzen:

- `serif` = eine Schriftart mit Serifen,
- `sans-serif` = eine Schriftart ohne Serifen,
- `cursive` = eine Schriftart für Schreibschrift,
- `fantasy` = eine Schriftart für ungewöhnliche Schrift,
- `monospace` = eine Schriftart mit dicktengleichen Zeichen.

Es empfiehlt sich, solche generischen Schriftarten als letzte Angabe einer Wertzuweisung an `font-family` zu notieren – wie im ersten der obigen Beispiele. Damit bieten Sie dem Browser

die Chance, eine Schriftart auszuwählen, die zumindest vom Typ her der gewünschten entspricht, falls die gewünschte nicht angezeigt werden kann.



### font-style (Schriftstil)

Schriftstil bedeutet die Neigung der Schrift. Ein Beispiel:

```
<html><head><title>font-style</title>
<style type="text/css">
.kursiv { font-style:italic; }
.grosskursiv { font-style:italic; font-size:200%; }
</style>
<body bgcolor="#FFFFFF" text="#000000">

<p><span class="kursiv">
    kleiner Beispieltext mit mit Schriftstil italic
</span><br>
<span class="grosskursiv">
    gro&szlig;er Beispieltext mit mit Schriftstil italic
</span></p>

</body></html>
```

Mit font-style: können Sie den Schriftstil bestimmen. Folgende Angaben sind möglich:

- italic = Schriftstil kursiv
- oblique = Schriftstil kursiv
- normal = normaler Schriftstil



### font-variant (Schriftvariante)

Als besondere Schriftvariante stehen Kapitälchen (kleine Großbuchstaben) zur Verfügung. Ein Beispiel:

```
<html><head><title>font-variant</title></head>
<body bgcolor="#FFFFFF" text="#000000">

<p><span style="font-variant:normal">kleiner Beispieltext
mit Schriftvariante normal</span><br>
<span style="font-variant:normal; font-size:200%">gro&szlig;er
Beispieltext mit Schriftvariante normal</span></p>

<p><span style="font-variant:small-caps">kleiner Beispieltext
mit Schriftvariante small-caps</span><br>
<span style="font-variant:small-caps; font-size:200%">gro&szlig;er
Beispieltext mit Schriftvariante scmall-caps</span></p>

</body></html>
```



16.2: Bei Kapitälchen verwendet der Browser durchgehend Großbuchstaben.

Mit `font-variant`: können Sie die Schriftvariante bestimmen. Folgende Angaben sind möglich:

- `small-caps` = Kapitälchen
- `normal` = normale Schriftvariante



### **font-size (Schriftgröße)**

Schriftgröße ist die Darstellungsgröße der Schrift. Ein Beispiel:

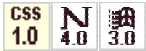
```
<html><head><title>font-size</title></head>
<body bgcolor="#FFFFFF" text="#000000">
<p>
<span style="font-size:6pt">Beispieltext mit Schriftgrö&ouml;&szlig;e 6pt</span><br>
<!-- usw. -->
</p></body></html>
```

Mit `font-size`: können Sie die Schriftgröße bestimmen. Erlaubt ist eine numerische Angabe. Auch Prozentangaben sind erlaubt. Prozentwerte beziehen sich auf die Schriftgröße des Eltern-elements. Alternativ zu numerischen Angaben sind auch folgende »ungenau« Angaben möglich:

- `xx-small` = winzig
- `x-small` = sehr klein
- `small` = klein
- `medium` = mittel
- `large` = groß
- `x-large` = sehr groß
- `xx-large` = riesig
- `smaller` = sichtbar kleiner als normal
- `larger` = sichtbar größer als normal

**Beachten Sie:** Sie können die Angabe zur Schriftgröße mit der Angabe zur Zeilenhöhe kombinieren, indem Sie beide Angaben innerhalb der Eigenschaft `font` mit der folgenden Syntax notieren (Beispiel): `p { font:12pt/14pt }`. In dem Beispiel ist 12pt die Schriftgröße und 14pt die Zeilen-

höhe. Bei dieser Notationsweise bezeichnet die Zahl vor dem Schrägstrich stets die Schriftgröße und die Zahl hinter dem Schrägstrich stets die Zeilenhöhe.



### font-weight (Schriftgewicht)

Das Schriftgewicht bezeichnet die Dicke und Stärke einer Schrift. Ein Beispiel:

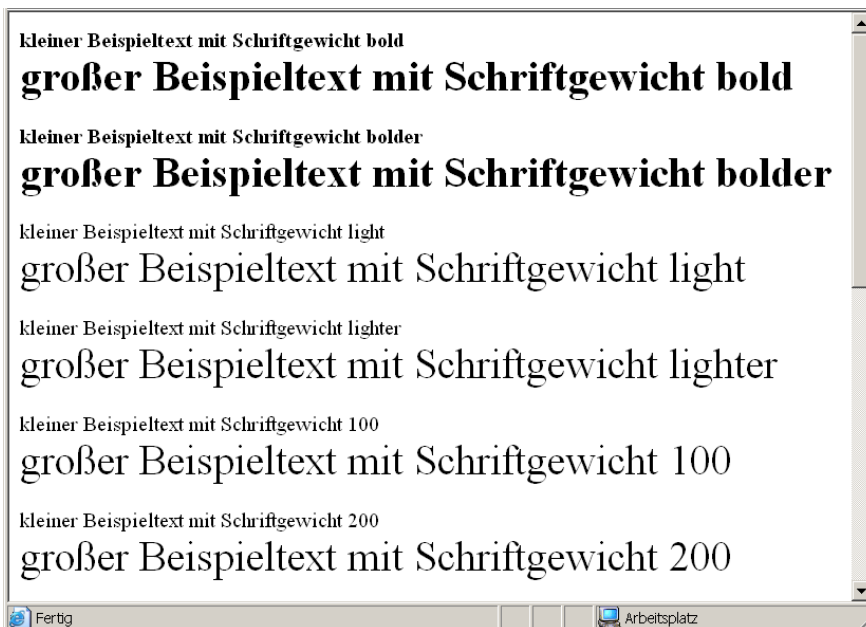
```
<html><head><title>font-weight</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p><span style="font-weight:bold">
kleiner Beispieltext mit Schriftgewicht bold
</span><br>
<span style="font-weight:bold; font-size:200%">
großzügiger Beispieltext mit Schriftgewicht bold
</span></p>

<!--usw.-->

<p><span style="font-weight:100">
kleiner Beispieltext mit Schriftgewicht 100
</span><br>
<span style="font-weight:100; font-size:200%">
großzügiger Beispieltext mit Schriftgewicht 100
</span></p>

<!--usw.-->
</body></html>
```



16.3: Die Windows-Standardschrift Times Roman unterstützt nicht alle Schriftgewichte.

Mit `font-weight`: können Sie das Schriftgewicht bestimmen. Folgende Angaben sind möglich:

- `bold` = **fett**
- `bolder` = **extra-fett**
- `lighter` = **dünn**
- 100, 200, 300, 400, 500, 600, 700, 800, 900 = **extra-dünn** (100) bis **extra-fett** (900)
- `normal` = normales Schriftgewicht

Bei den numerischen Werten entspricht die Angabe 500 dem im DTP-Bereich üblichen Begriff *medium*, und die Angabe 700 entspricht dem Begriff *bold*.

**Beachten Sie:** Wohl kaum eine installierte Schriftart unterstützt alle erlaubten Angaben zum Schriftgewicht.

### **CSS 2.0** *font-stretch (Schriftlaufweite)*

Die Schriftlaufweite gibt an, wie schmal oder breit eine Schrift erscheint. Ein Beispiel:

```
<html><head><title>font-stretch</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>
<span style="font-stretch:wider">
  kleiner Beispieltext mit Laufweite wider
</span><br>
<span style="font-stretch:wider; font-size:200%">
  gro&szlig;er Beispieltext mit Laufweite wider
</span></p>

<p>
<span style="font-stretch:narrower">
  kleiner Beispieltext mit Laufweite narrower
</span><br>
<span style="font-stretch:narrower; font-size:200%">
  gro&szlig;er Beispieltext mit Laufweite narrower
</span></p>

<!--usw.-->
</body></html>
```

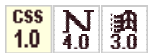
Mit `font-stretch`: können Sie die Laufweite der Schrift bestimmen. Folgende Angaben sind möglich:

- `wider` = **weiter als normal**
- `narrower` = **enger als normal**
- `condensed` = **gedrängt**
- `semi-condensed` = **halb gedrängt**



- extra-condensed = stark gedrängt
- ultra-condensed = extrastark gedrängt
- expanded = geweitet
- semi-expanded = halb geweitet
- extra-expanded = stark geweitet
- ultra-expanded = extrastark geweitet
- normal = normale Laufweite

**Beachten Sie:** Diese Eigenschaft wird von den verbreiteten Browsern noch nicht interpretiert.



### font (Schrift allgemein)

Diese Eigenschaft ist eine Zusammenfassung der folgenden fakultativen Einzelangaben: font-family, font-style, font-variant, font-size, font-weight, line-height. Ein Beispiel:

```
<html><head><title>font</title>
<style type="text/css">
#Text01 { font:bold 12pt Times; }
#Text02 { font:italic 1cm Helvetica; }
#Text03 { font:small-caps 14pt Verdana; }
</style>
<body bgcolor="#FFFFFF" text="#000000">

<p id="Text01">Beispieltext mit Schrift: bold 12pt Times</p>
<p id="Text02">Beispieltext mit Schrift: italic 1cm Helvetica</p>
<p id="Text03">Beispieltext mit Schrift: small-caps 14pt Verdana</p>

</body></html>
```



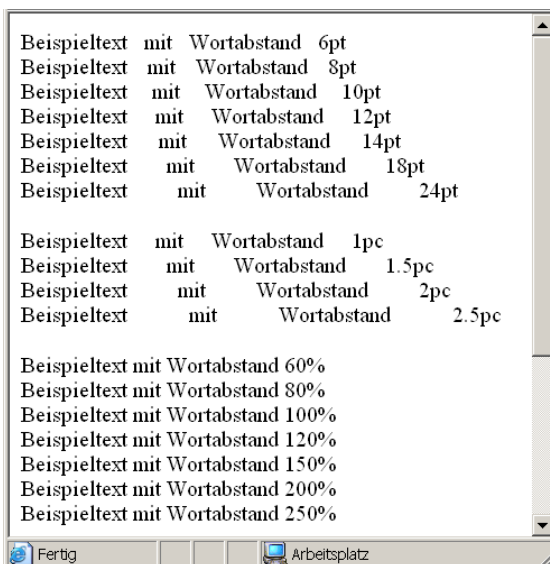
16.4: Mit font: kombinieren Sie die verschiedenen Schriftangaben.

Mit font: können Sie verschiedene Schriftformatierungen mischen. Erlaubt sind die üblichen Wertangaben zu den erlaubten Eigenschaften, die in font: zusammengefasst sind. Die Reihenfolge der Angaben ist egal.

### **word-spacing (Wortabstand)**

Mit dieser Eigenschaft können Sie den Abstand zwischen Wörtern im Text bestimmen. Ein Beispiel:

```
<html><head><title>word-spacing</title>
</head><body bgcolor="#FFFFFF" text="#000000">
<p>
<span style="word-spacing:6pt">
  Beispieltext mit Wortabstand 6pt
</span><br>
<span style="word-spacing:8pt">
  Beispieltext mit Wortabstand 8pt
</span><br>
<!-- usw. -->
</p>
</body></html>
```



16.5: Der Abstand zwischen den Wörtern ist mit CSS-Angaben variabel.

Mit `word-spacing`: können Sie den Wortabstand bestimmen. Erlaubt ist eine numerische Angabe, jedoch keine Prozentangaben.

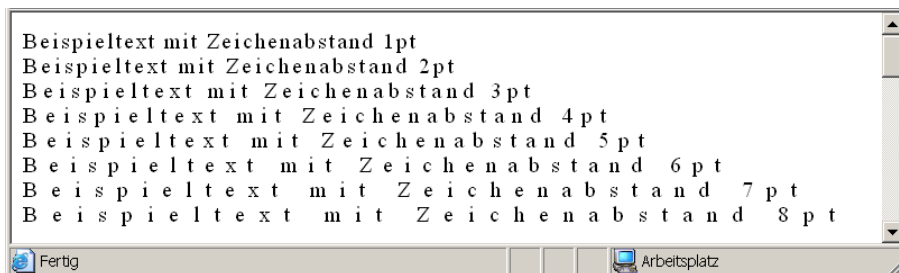
**Beachten Sie:** Diese Angabe wird noch nicht von allen Browsern interpretiert.

### **letter-spacing (Zeichenabstand)**

Mit dieser Eigenschaft können Sie den Abstand zwischen den Buchstaben bzw. Zeichen im Text bestimmen.

Ein Beispiel:

```
<html><head><title>letter-spacing</title>
</head><body bgcolor="#FFFFFF" text="#000000">
<p>
<span style="letter-spacing:1pt">
  Beispieltext mit Zeichenabstand 1pt
</span><br>
<span style="letter-spacing:2pt">
  Beispieltext mit Zeichenabstand 2pt
</span><br>
<!-- usw. -->
</p>
</body></html>
```



16.6: Den Abstand zwischen den einzelnen Buchstaben steuert letter-spacing:.

Mit letter-spacing: können Sie die Schriftgröße bestimmen. Erlaubt ist eine numerische Angabe, jedoch keine Prozentangabe.

**Beachten Sie:** Diese Angabe wird von Netscape 4.x noch nicht interpretiert.



### **text-decoration (Textdekoration)**

Mit dieser Eigenschaft können Sie zusätzliche Textformatierungen wie unterstrichenen oder durchgestrichenen Text erzwingen. Ein Beispiel:

```
<html><head><title>text-decoration</title>
<style type="text/css">
a:link { text-decoration:none; }
a:visited { text-decoration:line-through; }
a:hover { text-decoration:underline; }
a:active { text-decoration:underline; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

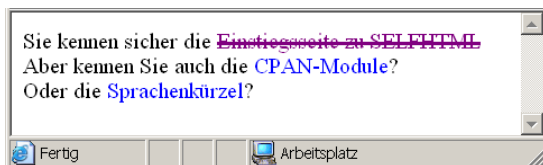
<p>
Sie kennen sicher die
<a href="../../index.htm">
  Einstiegsseite zu SELFHTML
```

```

</a><br>
Aber kennen Sie auch die
<a href="../../cgiperl/module/cpanmodule.htm">
  CPAN-Module
</a>?<br>
Oder die
<a href="../../diverses/sprachenkuerzel.htm">
  Sprachen&uuml;rzel
</a>?
</p>

</body>
</html>

```



16.7: Durchgestrichene oder blinkende Elemente sind mit CSS-Angaben kein Problem.

Mit `text-decoration`: können Sie die zusätzliche Formatierung bestimmen. Folgende Angaben sind möglich:

- `underline` = unterstrichen
- `overline` = überstrichen
- `line-through` = durchgestrichen
- `blink` = blinkend
- `none` = normal (keine Textdekoration)

**Beachten Sie:** Der MS Internet Explorer interpretiert die Angabe `blink` nicht. Netscape 4.x interpretiert die Angabe `overline` noch nicht, Netscape 6.x jedoch schon.

### **text-transform (Texttransformation)**

Mit dieser Eigenschaft können Sie in einem Textbereich Klein- oder Großbuchstaben oder Kapitälchen erzwingen, unabhängig davon, wie die einzelnen Buchstaben tatsächlich in der Datei stehen. Ein Beispiel:

```

<html><head><title>text-transform</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>
<span style="text-transform:capitalize">
  kleiner Beispielttext mit Texttransformation capitalize
</span><br>

```

```
<span style="text-transform:capitalize; font-size:200%">
  groß;er Beispieltext mit Texttransformation capitalize
</span></p>

<!--usw.-->

</body>
</html>
```



16.8: Der Internet Explorer unterstützt auch in Version 6 die Angabe `capitalize` nicht.

Mit `text-transform`: können Sie die Transformation bestimmen. Folgende Angaben sind möglich:

- `capitalize` = Wortanfänge als Großbuchstaben
- `uppercase` = nur Großbuchstaben
- `lowercase` = nur Kleinbuchstaben
- `none` = normal (keine Texttransformation)

**Beachten Sie:** Der MS Internet Explorer bis zur Version 6 interpretiert die Angabe `capitalize` noch nicht.



### **color (Textfarbe)**

Mit `color`: können Sie die Textfarbe bestimmen. Erlaubt sind CSS-üblichen Farbangaben. Ein Beispiel:

```
<html><head><title>color</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>
<span style="color:blue">
  kleiner Beispieltext mit Farbe <b>blue</b>
</span><br>
```

```
<span style="color:blue;font-size:200%">
  gro&szlig;er Beispieltext mit Farbe <b>blue</b>
</span></p>

<!--usw.-->
</body></html>
```

### **text-shadow (Textschatten)**

Mit `text-shadow`: können Sie einen Textschatten erzwingen. Erlaubt sind die CSS-üblichen Farbangaben oder der Wert `none` für »keinen Textschatten«. Ein Beispiel:

```
<html><head><title>text-shadow</title>
<style type="text/css">
#rotschattig { text-shadow:black; font-size:36pt; color:red; }
#blauschattig { text-shadow:black; font-size:36pt; color:blue; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<p id="rotschattig">
  Gro&szlig;er roter Text mit schwarzem Schatten
</p>
<p id="blauschattig">
  Gro&szlig;er blauer Text mit schwarzem Schatten
</p>

</body></html>
```

**Beachten Sie:** Dieser Befehl, der zur Version 2.0 der CSS Stylesheets gehört, wird bislang von keinem der verbreiteten Browser interpretiert.

## 16.2 Schriftformatierung mit Schriftartendatei

### **@font-face, src, font-family (Schriftformatierung mit Schriftartendatei)**

Mit dieser Eigenschaft können Sie eine Schriftart definieren und dabei direkt die Datenressourcen der gewünschten Schriftarten ansprechen, also bestimmte Schriftartendateien, oder zumindest mit Hilfe spezieller Befehle die Charakteristika der gewünschten Schriftart exakt beschreiben. Beim Definieren der Schriftart vergeben Sie einen Namen für die Schriftart. Unter diesem Namen können Sie sie anschließend mit der Eigenschaft `font-family` verwenden. Ein Beispiel:

```
<html><head><title>font-face</title>
<style type="text/css">
@font-face { font-family:Garamond; src:url(garamond.eot), url(garamond.pfr); }
</style>
</head>
<body>
<p style="font-family:Garamond">Text in Garamond</p>
</body></html>
```

Mit `@font-face` starten Sie die Angabe einer exakten Schriftart in einem `style`-Bereich. Dahinter folgen, in geschweiften Klammern, die Daten. Die beiden wichtigsten sind der Schriftartenname `font-family` und der URI der Schriftartendatei `src:url(Datei)`.

Den Schriftartennamen vergeben Sie mit der Angabe `font-family`: innerhalb des `@font-face`-Bereichs. Unter diesem Namen können Sie die Schriftart in HTML verwenden. Im obigen Beispiel gibt es einen Textabsatz, der mit Hilfe einer Stylesheet-Angabe die zuvor definierte Schriftart erzwingt.

Die Schriftartendatei, die Sie für die zugehörigen `font-family`-Angaben bestimmen, geben Sie mit `src:url(Datei)` an. Wenn sich die Schriftartendatei im gleichen Verzeichnis befindet wie die HTML-Datei, genügt die Angabe des Dateinamens. Andernfalls müssen Sie die genaue Lage der Schriftartendatei entweder durch eine relative Pfadangabe oder durch eine absolute Adressierung vom Typ `http://...` angeben.

Im obigen Beispiel sehen Sie, dass gleich zwei URI-Angaben notiert sind. Einmal wird eine `eot`-Datei als Schriftartenquelle angegeben, einmal eine `pfr`-Datei. Beide Formate sind browser-spezifisch: `eot`-Dateien sind Schriftartendateien, die der MS Internet Explorer (ab Version 4.0) als Web-Schriftarten erkennt; `pfr`-Dateien sind Schriftartendateien, die Netscape (ab Version 4.0) erkennt. Wenn Sie Ihre gewünschte Schriftart in beiden Formaten anbieten können, können Sie beide Formate so einbinden wie im obigen Beispiel.

Einzelheiten zu `.eot`- und `.pfr`-Schriftarten und wie Sie diese selbst erstellen können, finden Sie im Kapitel 23.4 über *Internationalisierung* im Abschnitt *Downloadbare Schriftarten*.

### **CSS 2.0** Schriftformate eingrenzen

Wenn Sie eine Schriftart mit einer Schriftartendatei definieren, können Sie bei der Definition auch Angaben zu den möglichen Formateigenschaften der Schrift machen, wie z.B. zu Schriftstil (`font-style`), Schriftvariante (`font-variant`), Schriftgröße (`font-size`) oder Schriftgewicht (`font-weight`). So können Sie dem WWW-Browser bereits in der HTML-Datei mitteilen, welche Formateigenschaften die Schriftartendatei unterstützt. Der Browser überträgt die Schriftartendatei in diesem Fall nur dann zum Anwender, wenn Formatierungen in der HTML-Datei die Übertragung sinnvoll machen. Ein Beispiel:

```
<html><head><title>font-face</title>
<style type="text/css">
  @font-face { font-family:Garamond; src:url(garamond.eot); font-
size:10pt,12pt,14pt; }
  @font-face { font-family:Garamond; src:url(garamond.pfr); font-
size:10pt,12pt,14pt; }
</style>
</head><body>
<p style="font-family:Garamond; font-size:48pt">Text</p>
<!-- Datei wird nicht übertragen -->
</body></html>
```

Mit den dafür üblichen Stylesheet-Befehlen können Sie bei der Definition einer Schriftart mit Schriftartendatei einschränkende Angaben zur Schrift machen. Folgende Besonderheiten sollten Sie dabei beachten:

- Bei `font-style` können Sie auch mehrere erlaubte Angaben machen. Trennen Sie diese Angaben durch Kommata, also beispielsweise: `font-style:normal,italic;.`
- Bei `font-variant` können Sie auch mehrere erlaubte Angaben machen. Trennen Sie diese Angaben durch Kommata, also beispielsweise: `font-style:normal,small-caps;.`
- Bei `font-weight` sind die Angaben `bolder` und `lighter` nicht zulässig. Außerdem können Sie auch mehrere erlaubte Angaben machen. Trennen Sie diese Angaben durch Kommata, also beispielsweise: `font-weight:normal,bold,900;.`
- Bei `font-size` sind nur absolute Größenangaben erlaubt, wie etwa `font-size:18pt`. Relative Größenangaben wie Prozentangaben oder `em`-Angaben sind nicht erlaubt. Außerdem können Sie auch mehrere Angaben machen. Trennen Sie diese Angaben durch Kommata, also beispielsweise: `font-size:12pt,14pt,16pt;.`

#### **CSS 2.0** *unicode-range (Unicode-Bereich eingrenzen)*

Wenn Sie eine Schriftart mit einer Schriftartendatei definieren, können Sie bei der Definition den Zeichenbereich einschränken. So können Sie dem WWW-Browser bereits in der HTML-Datei mitteilen, welchen Zeichenbereich die Schriftart unterstützt. Der Browser überträgt die Schriftartendatei in diesem Fall nur dann zum Anwender, wenn alle Zeichen eines Textes, der mit der Schriftart formatiert wird, innerhalb des definierten Unicode-Bereichs liegen. Dabei gilt das Unicode-System als Grundlage (siehe Anhang). Ein Beispiel:

```
<html><head><title>unicode-range</title>
<style type="text/css">
  @font-face { font-family:System; src:url(system.eot); unicode-range:U+0000-007F; }
  @font-face { font-family:System; src:url(system.pfr); unicode-range:U+0000-007F; }
</style>
</head>
<body>
<tt style="font-family:System">Text mit ß und Ü</tt>
<!-- Datei wird nicht übertragen -->
</body>
</html>
```

Mit `unicode-range`: können Sie den Zeichenbereich der Schriftartendatei vor dem Download bekannt machen. Im obigen Beispiel wird der Bereich `U+0000-007F` definiert. Das entspricht dem ASCII-Zeichensatz. Wenn nun in einem Bereich, der diese Schriftart benutzt, beispielsweise Umlaute oder andere Zeichen vorkommen, die nicht im ASCII-Zeichensatz vorkommen, wird die Schriftartendatei nicht heruntergeladen.

Unicode-Angaben beginnen immer mit `U+`. Dahinter folgt eine Angabe des gewünschten Zeichenwerts in hexadezimaler Form. Um einen Bereich zu definieren, der aus mehreren Unicode-Zeichen besteht, können Sie als Platzhalterzeichen (Wildcard) ein Fragezeichen `?` benutzen. Bei einer Angabe wie `unicode-range:U+00??` bewirken die zwei hintereinander stehenden Fragezeichen am Ende der Angabe, dass der gewünschte Unicode-Bereich  $16 \text{ hoch } 2$ , also 256 Zeichen ab dem Zeichen `U+00` beträgt. Da laut Unicode-System bei `U+0000` die Zeichen des so genannten Western-Latin-1-Zeichensatzes beginnen, der aus 256 Zeichen besteht, deckt die Angabe in dem



Beispiel genau diesen Zeichensatz ab. Wenn Sie einen beliebigen Unicode-Bereich bestimmen möchten, können Sie dies mit einer Angabe tun wie z.B. U+AC00-D7FF.



### Weitere Angaben zur Schriftquelle

Die Version 2.0 der CSS-Sprache stellt einige Stylesheet-Angaben bereit, um Schriftartenquellen möglichst vielfältig definierbar und auffindbar zu machen. Ein Beispiel:

```
<style type="text/css">
@font-face { font-family:Kino; src:local(Kino MT), url(kino.ttf) format(TrueType);
}
@font-face { font-family:MeineArt; url(http://www.xy.de/cgi-bin/myfont.pl)
format(intellifont) }
@font-face { font-family:Alabama; url(alab.fon); panose-1: 2 4 5 2 5 4 5 9 3 3; }
</style>
```

Sie können bewirken, dass der WWW-Browser zuerst auf dem Rechner des Anwenders nach der gewünschten Schriftart sucht und eine auf dem Server bereitliegende Schriftart nur dann herunterlädt, wenn die Schriftart lokal nicht verfügbar ist. Dazu notieren Sie bei der Quellenangabe der Schriftartendatei (`src:`) zuerst `local` und dahinter in Klammern den gewünschten Schriftartennamen. Für den Fall, dass der Browser diese Schriftart beim Anwender nicht findet, können Sie, durch ein Komma getrennt, dahinter die Schriftartendatei angeben, die Sie auf Ihren WWW-Seiten mit anbieten – im Beispiel `url(kino.ttf)`.

Sie können ferner Angaben zum Format der Schriftart machen. Der WWW-Browser lädt die Schriftartendatei in diesem Fall nur, wenn sie auf dem Rechner des Anwenders ausführbar ist. Dazu notieren Sie, durch ein Leerzeichen getrennt von den Angaben zu `src:local()` und/oder `src:url()`, die Angabe `format()`. Innerhalb der Klammern können Sie das Format der Schriftartendatei angeben. Im obigen Beispiel wird eine TrueType-Schriftart angegeben. Dieses Schriftartenformat ist nur in der Windows-Welt verbreitet. Anwender, die z.B. einen Macintosh oder einen Amiga-Rechner einsetzen, haben von einer solchen Datei nichts. Ein WWW-Browser, der diese Angaben kennt, braucht also eine solche Datei nicht downzuladen.

Es gibt weitere Möglichkeiten, die Schriftart genau einzugrenzen:

- Mit `panose-1:`, gefolgt von 10 durch Leerzeichen getrennten Zahlen, können Sie eine Schriftart nach dem Panose-1-System bestimmen. Beispiel: `panose-1: 2 4 5 2 5 4 5 9 3 3;`
- Mit `stembv:` und `stemh:` können Sie die vertikale und horizontale Stammbreite der Glyphen angeben.
- Mit `units-per-em:` können Sie die Anzahl der Einheiten pro `em`-Maßeinheit angeben. Üblich sind die Werte 250 (bei Intellifont), 1000 (bei Type-1) oder 2048 (bei TrueType). Wenn Sie `stembv:` und/oder `stemh:` angeben, müssen Sie auch `units-per-em:` angeben.
- Mit `slope:` können Sie den vertikalen Winkel der Glyphen angeben.
- Mit `cap-height:` können Sie die Höhe der Großbuchstaben der Glyphen angeben.
- Mit `x-height:` können Sie die Höhe der Kleinbuchstaben der Glyphen angeben.
- Mit `ascent:` können Sie die maximale Höhe der akzentlosen Glyphen angeben.

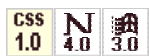
- Mit `descent`: können Sie die maximale Tiefe der akzentlosen Glyphen angeben.
- Mit `definition-src:url()` können Sie eine Datei mit Schriftartendefinitionen angeben.
- Mit `definition-src:url()` können Sie eine Datei mit Schriftartendefinitionen angeben.
- Mit `baseline`: können Sie die untere Grundlinie der Glyphen angeben.
- Mit `centerline`: können Sie die zentrale Grundlinie der Glyphen angeben.
- Mit `mathline`: können Sie die mathematische Grundlinie der Glyphen angeben.
- Mit `topline`: können Sie die obere Grundlinie der Glyphen angeben.

## 16.3 Ausrichtung und Absatzkontrolle

### Allgemeines zu Ausrichtung und Absatzkontrolle

Für Textblöcke oder Textabsätze gibt es absatztypische Eigenschaften. Absätze können beispielsweise links- oder rechtsbündig ausgerichtet sein. Bei Tabellenzellen oder anderen Elementen, die eine berechnete oder definierte Höhe haben können und mehr Platz lassen, als der Text darin einnimmt, wird häufig vertikale Ausrichtung oben, mittig oder untenbündig erforderlich.

Zu den typischen Eigenschaften der Absatzformatierung gehören ferner die Merkmale Einzug und Zeilenhöhe. Und schließlich auch noch die Angabe, wie der Fließtext innerhalb des Absatzes umgebrochen werden soll: ob gar nicht, automatisch, oder so, wie er im Editor eingegeben wurde.



### *text-indent (Texteinrückung)*

Sie können für einen mehrzeiligen Fließtext bestimmen, dass die erste Zeile eingerückt wird. Dieses Gestaltungselement ist in Büchern und Zeitschriften sehr verbreitet. Mit `text-indent`: können Sie eine Einrückung für die erste Zeile bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>text-indent</title>
</head><body bgcolor="#FFFFFF" text="#000000">

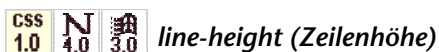
<p style="text-indent:0.5cm;">Dieser Textabsatz hat eine Texteinrückung von
0.5cm. Texteinrückungen dieser Art sind an die Ästhetik von Büchern
oder Zeitungen angelehnt, wo häufig mit dieser Formatierung gearbeitet wird.
Damit das auch wirklich sichtbar wird, enthält dieser Textabsatz
schließlich noch diesen überflüssigen Satz, der sich aber
hoffentlich wenigstens flüssig liest.</p>

<!--usw.-->
</body></html>
```



16.9: Verschiedene Formen, einen Absatz mit text-indent: einzurücken.

**Beachten Sie:** Mit einem negativen Wert für text-indent bewirken Sie eine Textausrückung in der ersten Fließtextzeile.



### line-height (Zeilenhöhe)

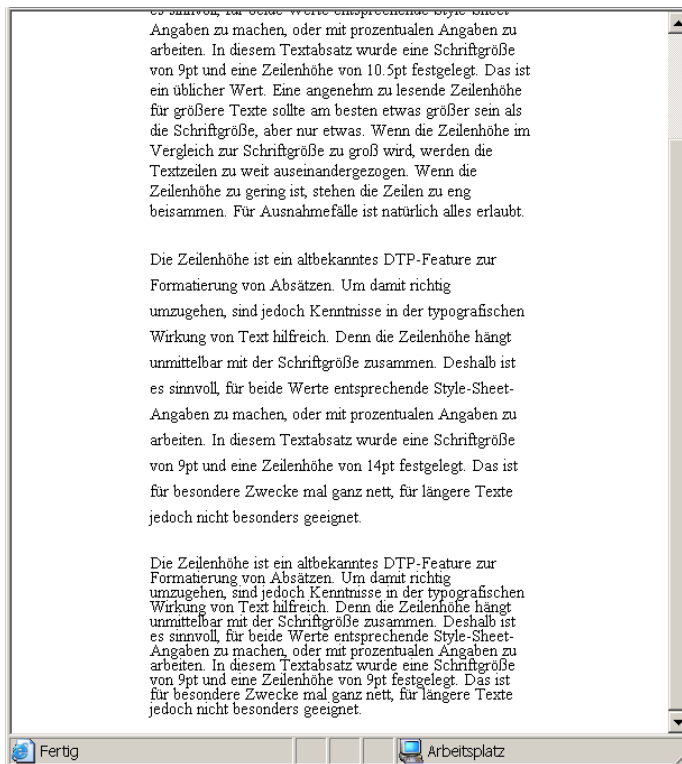
Sie können für Elemente mit viel Fließtext die Zeilenhöhe bestimmen. Diese Angabe ist vor allem in Verbindung mit einer Angabe zur Schriftgröße interessant. Ein Beispiel:

```
<html><head><title>line-height</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p style="line-height:10.5pt;font-size:9pt;margin-left:20%;margin-right:20%">
Die Zeilenhöhe ist ein altes bekanntes DTP-Feature zur Formatierung von
Absätzen. Um damit richtig umzugehen, sind jedoch Kenntnisse in der
typografischen Wirkung von Text hilfreich. ... </p>

</body></html>
```

Mit line-height: können Sie die Zeilenhöhe bestimmen. Erlaubt ist eine numerische Angabe. Auch Prozentangaben sind erlaubt. Prozentwerte beziehen sich dabei auf die Schriftgröße des Elements, für das die Zeilenhöhe bestimmt wird.



16.10: Drei Absätze mit unterschiedlicher Zeilenhöhe, aber identischer Schrift.

**Beachten Sie:** Bei erzwungener Zeilenhöhe ist es möglich, dass ein Browser dieser Angabe den Vorrang einräumt und Elemente abschneidet, die höher sind, beispielsweise eine im Fließtext referenzierte Grafik.



### **vertical-align (vertikale Ausrichtung)**

Sie können nebeneinander stehende Elemente mit unterschiedlicher Höhe, zum Beispiel Tabellenzellen in einer Tabellenzeile oder Textpassagen mit unterschiedlicher Schriftgröße innerhalb einer Zeile, im Verhältnis zueinander ausrichten. Ein Beispiel:

```
<html><head><title>vertical-align</title>
<style type="text/css">
.oben { vertical-align:top; background-color:#CCCCCC; }
.mittig { vertical-align:middle; background-color:#DDDDDD; }
.unten { vertical-align:bottom; background-color:#EEEEEE; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<table border="1"><tr>
<td height="200" class="oben"><b>Text oben</b></td>
<td height="200" class="mittig"><b>Text mittig</b></td>
```

```
<td height="200" class="unten"><b>Text unten</b></td>
</tr></table>

</body></html>
```

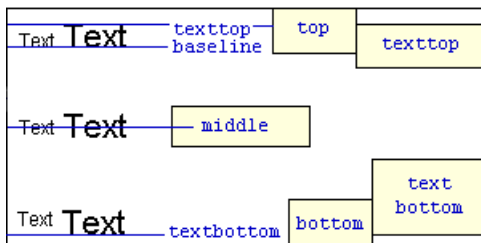


16.11: Der Text in den drei Tabellenzellen ist unterschiedlich ausgerichtet.

Mit `vertical-align`: können Sie die vertikale Ausrichtung bestimmen. Folgende Angaben sind möglich:

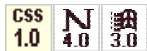
- `top` = obenbündig ausrichten
- `middle` = mittig ausrichten
- `bottom` = untenbündig ausrichten
- `baseline` = an der Basislinie ausrichten (oder untenbündig, wenn es keine Basislinie gibt)
- `sub` = tieferstellen (ohne die Schriftgröße zu reduzieren)
- `super` = höherstellen (ohne die Schriftgröße zu reduzieren)
- `text-top` = am oberen Schriftrand ausrichten
- `text-bottom` = am unteren Schriftrand ausrichten

Bei Fließtext ist auch eine prozentuale Angabe möglich. Die Ausrichtung orientiert sich dann an der elementeigenen Zeilenhöhe. Das folgende Bild veranschaulicht das Modell der vertikalen Ausrichtung:



16.12: So wirken sich die verschiedenen `text-align`-Angaben aus.

Die grauen Kästen stellen Kindelemente des großen Kastens dar, die frei stehenden schwarzen Texte stellen Zeicheninhalte des großen Kastens dar.



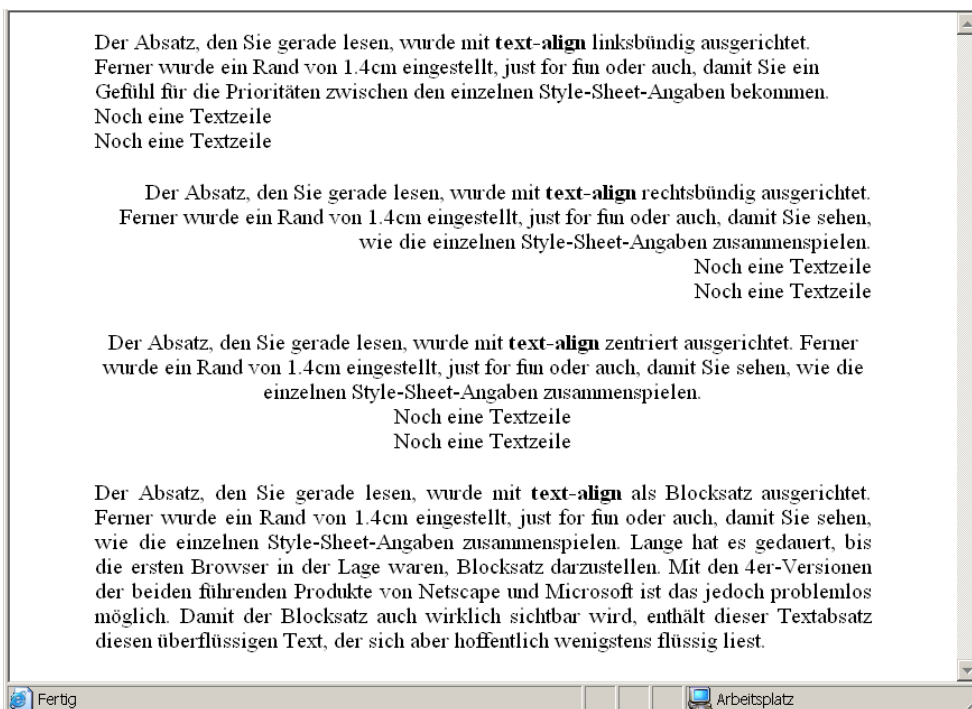
### text-align (horizontale Ausrichtung)

Sie können Textabsätze mit `text-align`: und andere Blockelemente mit Fließtext, z.B. Tabellenzellen, ausrichten. Ein Beispiel:

```
<html><head><title>text-align</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p style="text-align:left; margin-left:1.4cm; margin-right:1.4cm">Der Absatz, den
Sie gerade lesen, wurde mit <b>text-align</b> linksbündig ausgerichtet. Ferner
wurde ein Rand von 1.4cm eingestellt, just for fun oder auch, damit Sie sehen, wie
die einzelnen Stylesheet-Angaben zusammenspielen.<br>
Noch eine Textzeile<br>
Noch eine Textzeile</p>

<!-- usw. -->
</body></html>
```



16.13: Einige Absätze mit unterschiedlicher horizontaler Ausrichtung

Folgende Angaben sind möglich:

- `left` = linksbündig ausrichten (Voreinstellung)
- `center` = zentriert ausrichten
- `right` = rechtsbündig ausrichten
- `justify` = als Blocksatz ausrichten

**Beachten Sie:** Eine ehemals eingeführte CSS-Eigenschaft namens `alignment` ist mittlerweile aus dem CSS-Standard verschwunden. Verwenden Sie nur noch `text-align`.

### *white-space (Textumbruch)*

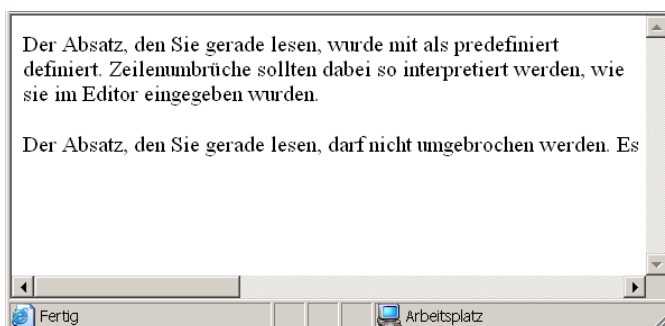
Sie können festlegen, wie der Zeilenumbruch innerhalb von Elementen mit Text erfolgen soll. Ein Beispiel:

```
<html><head><title>white-space</title>
<style type="text/css">
#editor      { white-space:pre; }
#langezeile  { white-space:nowrap; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<p id="editor">
Der Absatz, den Sie gerade lesen,
wurde mit CSS als predefiniert definiert.
    Zeilenumbrüche sollten dabei so interpretiert
        werden, wie sie im Editor eingegeben wurden.</p>

<p id="langezeile">Der Absatz, den Sie gerade lesen, darf nicht umgebrochen werden.
Es sollte kein automatischer Zeilenumbruch erfolgen. Bei diesem längeren Text
müssen Sie vermutlich quer scrollen, um alles zu lesen.</p>

</body></html>
```



16.14: Der obere Absatz ist als präformatiert deklariert, der untere als Bereich ohne Zeilenumbruch.

Mit `white-space`: können Sie das Verhalten für den Zeilenumbruch festlegen. Folgende Angaben sind möglich:

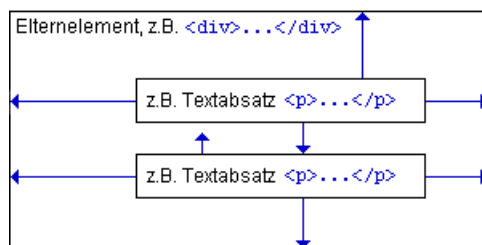
- `normal` = automatischer Zeilenumbruch (wie bei allen HTML-Tags außer `<pre>` und `<nowrap>`)
- `pre` = Zeilenumbruch wie im Editor eingegeben (so wie bei `<pre>...</pre>`)
- `nowrap` = Kein automatischer Zeilenumbruch, Umbruch möglich durch entsprechende HTML-Tags

**Beachten Sie:** Diese Angabe gehört zur CSS-Version 2.0. Der MS Internet Explorer interpretiert nur den Wert `nowrap`, nicht den Wert `pre`. Netscape 4.x interpretiert den Wert `pre`, nicht aber `nowrap`. Erst Netscape 6.x beherrscht alle Angaben.

## 16.4 Außenrand und Abstand

### Allgemeines zu Außenrand und Abstand

Außenrand oder Abstand bedeutet: erzwungener Leerraum zwischen dem aktuellen Element und seinem Elternelement oder Nachbarelement. Für ein `p`-Element, also einen Textabsatz etwa, der direkt innerhalb des `body`-Elements notiert ist, markieren Angaben zu linkem und rechten Außenrand seinen Abstand zu den Elementgrenzen des `body`-Elements. Wenn mehrere solcher `p`-Absätze aufeinander folgen, markieren Angaben zum Außenrand oben und unten den Abstand zwischen den Absätzen. Das folgende Bild veranschaulicht die Möglichkeiten, Außenränder und Abstände zu definieren:



16.15: Das Schema zeigt die verschiedenen Abstände zwischen Elementen.

Sinnvoll sind die hier beschriebenen CSS-Eigenschaften für alle HTML-Elemente, die einen eigenen Absatz erzeugen bzw. einen Block bilden, also etwa für `h[1-6]`, `p`, `blockquote`, `address` oder `pre`. Blockelemente sind aber auch HTML-Elemente wie `div`, `table`, `tr`, `th` und `td`. Auch auf `body` lassen sich die hier aufgelisteten Stylesheet-Angaben anwenden – in diesem Fall werden Ränder für den gesamten sichtbaren Körper der HTML-Datei definiert.

Mit negativen Werten, also Angaben wie beispielsweise `-18mm`, können Sie für besondere gestalterische Zwecke erreichen, dass sich Elemente überlappen.





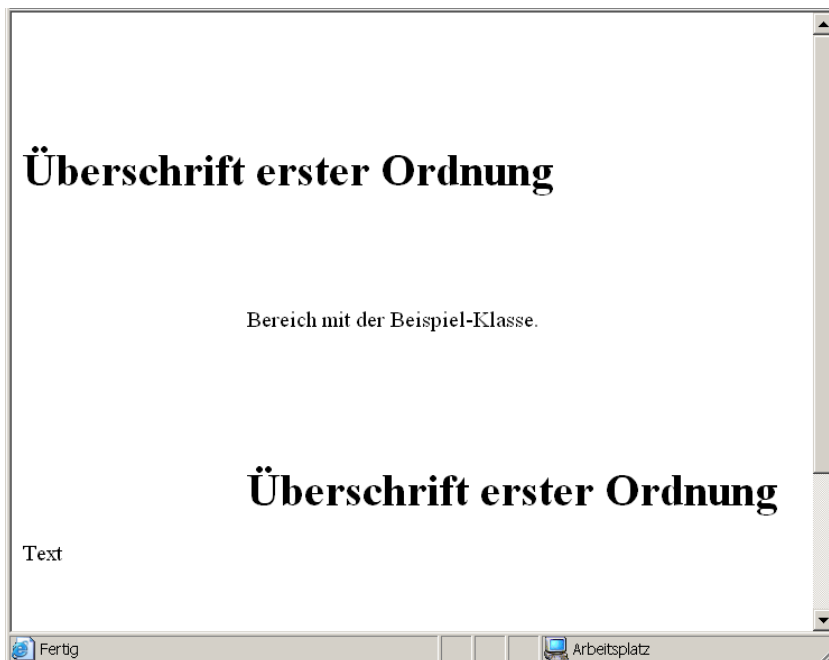
### **margin-top (Außenrand/Abstand oben)**

Sie können bestimmen, wie viel Abstand ein Element zu seinem Vorgängerelement oder zu seinem Elternelement nach oben hin hat. Ein Beispiel:

```
<html><head><title>margin-top</title>
<style type="text/css">
  h1,h2,h3 { margin-top:2.5cm; }
  div.beispiel { margin-top:100px; margin-left:200px; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<h1>Überschrift erster Ordnung</h1>
<div class="beispiel">
Bereich mit der Beispiel-Klasse.
<h1>Überschrift erster Ordnung</h1>
</div>

</body></html>
```



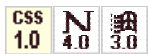
16.16: Verschiedene Elemente mit individuellen Abständen.

Mit `margin-top` können Sie den Rand/Abstand oben bestimmen. Erlaubt ist eine numerische Angabe. Im obigen Beispiel wird für Überschriften 1., 2. und 3. Ordnung ein vorausgehender Abstand von 2.5 cm definiert. Ferner wird eine Klasse `beispiel` für das `div`-Element definiert.

Solche Bereiche haben laut Definition im Beispiel 100 Pixel Abstand zu ihrem vorausgehenden Element (und 200 Pixel Abstand nach links, definiert mit `margin-left`).

Wenn das Vorgänger- oder Elternelement eine Angabe zu `margin-bottom` hat, gilt nur eine der beiden Angaben für den Gesamtabstand zwischen den beiden Elementen, und zwar diejenige, die den größten Abstand definiert. Wenn also ein erster Textabsatz `margin-bottom:10px` hat und ein folgender Textabsatz `margin-top:15px`, dann wird der Abstand zwischen beiden Absätzen auf 15px berechnet. Wenn eine der beiden Gegenangaben negativ ist, wird der tatsächliche Abstand zwischen den beiden Elementen aus dem Wert der positiven abzüglich der negativen Angabe berechnet.

**Beachten Sie:** Bei Verwendung im `body`-Element bestimmt `margin-top` den Abstand zum oberen Fensterrand. Einige frühe Netscape 4.x-Versionen beherrschen das Zusammenrechnen von Abstandsangaben oben und unten noch nicht und addieren beide Werte.



### ***margin-bottom (Außenrand/Abstand unten)***

Sie können bestimmen, wie viel Abstand ein Element zu seinem Nachfolgerelement nach unten hin hat – oder, wenn es das letzte Kindelement eines Elements ist – wie viel Abstand es zu dessen unterem Rand hin hat. Ein Beispiel:

```
<html><head><title>margin-bottom</title></head>
<body bgcolor="#FFFFFF" text="#000000">

<p style="margin-bottom:2cm">Dieser Textabsatz hat einen
Abstand von 2cm zum darauffolgenden.</p>

<p style="margin-bottom:18pt">Dieser Textabsatz hat einen
Abstand von 18pt zum darauffolgenden.</p>

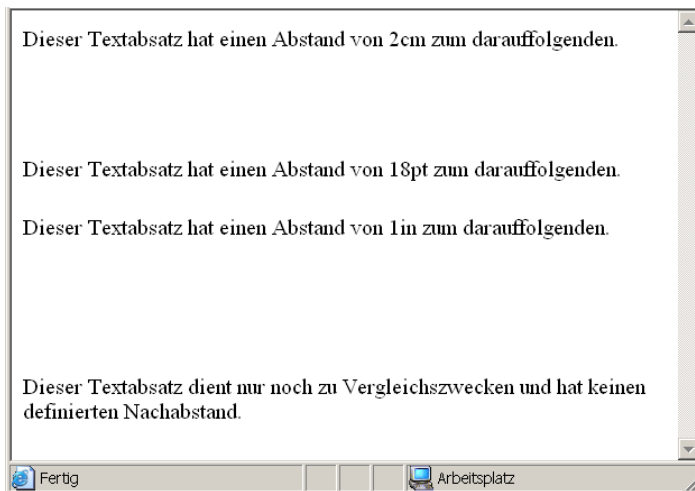
<p style="margin-bottom:1in">Dieser Textabsatz hat einen
Abstand von 1in zum darauffolgenden.</p>

<p>Dieser Textabsatz dient nur noch zu Vergleichszwecken und hat keinen
definierten Nachabstand.</p>

</body></html>
```

Mit `margin-bottom` können Sie den Rand/Abstand unten stimmen. Erlaubt ist eine numerische Angabe. Für Hinweise zu addierenden Angaben für `margin-bottom` und `margin-top` bei aufeinander folgenden Elementen siehe `margin-top`.

**Beachten Sie:** Bei Verwendung im `body`-Element bestimmt `margin-bottom` den Abstand zum unteren Fensterrand. Bei Netscape 4.x hat die Angabe `margin-top` (für den Abstand oben) in jedem Fall Vorrang vor der Angabe `margin-bottom` (für den Abstand unten), weshalb dort das intelligente Addieren zwischen Angaben zu beiden Eigenschaften nicht so funktioniert, wie es sollte.



16.17: Mehrere Absätze mit unterschiedlichen Abständen nach unten.

**margin-left (Außenrand/Abstand links)**

Sie können den linken Abstand/Rand eines Elements festlegen. Wenn das Element einen linken Nachbarn hat (etwa bei zwei nebeneinander liegenden Grafiken die zweite der beiden), dann bedeutet die Angabe den Abstand zum links davon liegenden Element. Wenn das Element keinen linken Nachbarn hat (etwa ein normaler Textabsatz), dann bedeutet die Angabe den Rand relativ zum Elternelement (z.B. zum Seitenrand, der für das `body`-Element festgelegt ist). Wenn kein übergeordnetes Elternelement mehr da ist (also beim `body`-Tag selbst), dann bedeutet die Angabe den linken Abstand des Dokumentinhalts zum Anzeigefenster. Ein Beispiel:

```
<html><head><title>margin-left</title>
<style type="text/css">
  body { margin-left:100px; }
  p,div,table,ul,ol { margin-left:30px; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

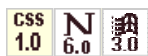
<h1>Eine &Uuml;berschrift erster Ordnung</h1>
<p>Ein Textabsatz, nochmals weiter eingezogen.</p>
<p>Noch einer davon.</p>
<h2>Eine &Uuml;berschrift zweiter Ordnung</h2>
<ul>
<li>Ein Listenpunkt</li>
<li>Noch ein Listenpunkt</li>
</ul>

</body></html>
```

Mit `margin-left`: können Sie den Rand/Abstand links bestimmen. Erlaubt ist eine numerische Angabe. Im obigen Beispiel wird zunächst ein globaler linker Seitenrand von 100 Pixeln definiert, indem diese Angabe dem `body`-Element zugewiesen wird. Für Fließtextabsätze, Bereiche,

Tabellen, und Listen wird zusätzlich ein linker Rand von 30 Pixeln definiert, sodass sich bei diesen Elementen insgesamt ein linker Rand von 135 Pixeln zum linken Fensterrand ergibt. Überschriften etwa, für die nichts weiter festgelegt ist, erscheinen dann links ausgerückt. Bei Nachbarelementen oder verschachtelten Elementen, bei denen Angaben zu `margin-left` und `margin-bottom` aufeinander treffen, werden beide Angaben in jedem Fall addiert.

**Beachten Sie:** Wenn Sie beispielsweise zwei Grafiken nebeneinander referenzieren und dabei mit Werten für `margin-left` oder `margin-right` arbeiten, reagiert Netscape 4.x zum Teil sehr unterschiedlich und nicht besonders nachvollziehbar.



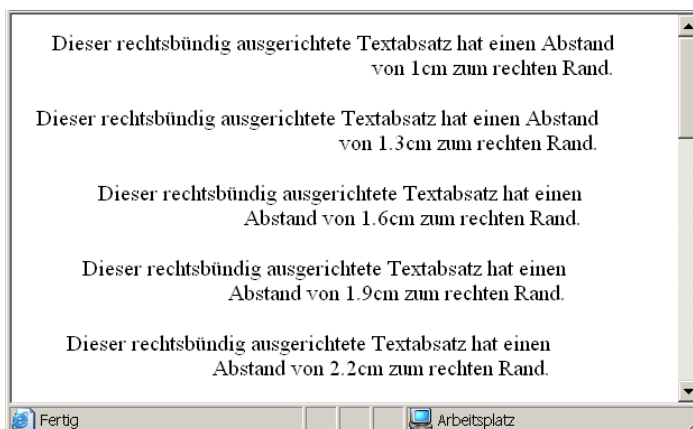
### *margin-right (Außenrand/Abstand rechts)*

Sie können den rechten Abstand/Rand eines Elements festlegen. Wenn das Element einen rechten Nachbarn hat (etwa bei zwei nebeneinander liegenden Grafiken die erste der beiden), dann bedeutet die Angabe den Abstand zum rechts davon liegenden Element. Wenn das Element keinen rechten Nachbarn hat (etwa ein normaler Textabsatz), dann bedeutet die Angabe den Rand relativ zum Elternelement (z.B. zum Seitenrand, der für das `body`-Element festgelegt ist.). Wenn kein übergeordnetes Elternelement mehr da ist (also beim `body`-Element selbst), dann bezieht sich die Angabe auf den rechten Abstand des Dokumentinhalts zum Anzeigefenster. Mit `margin-right`: können Sie den Rand/Abstand rechts bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>margin-right</title>
</head><body bgcolor="#FFFFFF" text="#000000">

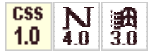
<p align="right" style="margin-right:1cm">Dieser rechtsbündig ausgerichtete
Textabsatz hat einen Abstand von 1cm zum rechten Rand.</p>
<p align="right" style="margin-right:1.3cm">Dieser rechtsbündig ausgerichtete
Textabsatz hat einen Abstand von 1.3cm zum rechten Rand.</p>

<!-- usw. -->
</body></html>
```



16.18: Absätze mit ansteigendem Abstand zum rechten Rand.

**Beachten Sie:** Netscape 4.x ignoriert `margin-right`.



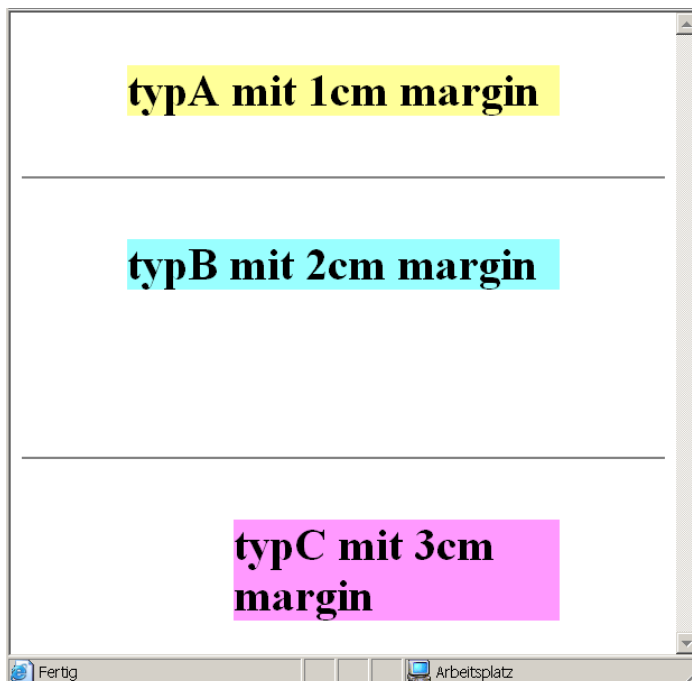
### **margin (Außenrand/Abstand allgemein)**

Diese Angabe ist eine Zusammenfassung der folgenden fakultativen Einzelangaben: `margin-top`, `margin-bottom`, `margin-left`, `margin-right`. Ein Beispiel:

```
<html><head><title>margin</title>
<style type="text/css">
  .typA { margin: 1cm 2cm;
          background-color:#FFFF99; }
  .typB { margin: 1cm 2cm 3cm;
          background-color:#99FFFF; }
  .typC { margin: 1cm 2cm 3cm 4cm;
          background-color:#FF99FF; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<div class="typA"><h1>typA mit 1cm margin</h1></div>
<hr noshade="1">
<div class="typB"><h1>typB mit 2cm margin</h1></div>
<hr noshade="1">
<div class="typC"><h1>typC mit 3cm margin</h1></div>

</body></html>
```



16.19: Absätze mit einer Kombination aus verschiedenen Randabständen.

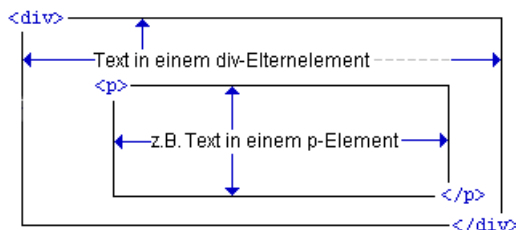
Mit `margin`: können Sie einen einheitlichen Rand/Abstand für oben, unten, links und rechts bestimmen. Erlaubt sind ein bis vier numerische Angaben. Eine Angabe bedeutet: Alle vier Ränder des Elements erhalten den gleichen Außenabstand.

- Bei mehreren Angaben werden die Angaben intern nach der angenommenen Reihenfolge für 1=oben, 2=rechts, 3=unten, 4=links interpretiert.
- Zwei Angaben: Die erste Angabe bedeutet den Abstand für oben und unten, die zweite Zahl den Abstand für rechts und links.
- Drei Angaben: Die erste Angabe bedeutet den Abstand für oben, die zweite den Abstand für rechts und links und die dritte den Abstand für unten.
- Vier Angaben: Die erste Angabe bedeutet den Abstand für oben, die zweite den Abstand für rechts, die dritte den Abstand für unten und die vierte den Abstand für links.

## 16.5 Innenabstand

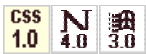
### Allgemeines zum Innenabstand

Innenabstand bedeutet: erzwungener Leerraum zwischen dem Inhalt eines Elements zu seinem eigenen Elementrand, also z.B. zwischen dem Text eines Elements und dem Rand dieses Elements. Das folgende Bild zeigt, wie Innenabstände zu verstehen sind (die blauen Pfeile stehen für definierte Innenabstände):



16.20: Das Schema zeigt die verschiedenen Innenabstände um ein Element.

Sinnvoll sind die hier beschriebenen CSS-Eigenschaften für alle HTML-Elemente, die einen eigenen Absatz erzeugen bzw. einen Block bilden, also etwa für `h[1-6]`, `p`, `blockquote`, `div`, `address` oder `pre`. Sinnvoll sind die Angaben zum Innenabstand ferner, wenn Sie außerdem CSS-Eigenschaften für Rahmen (Kapitel 16.6) oder für Farben oder Grafiken zum Hintergrund eines Elements (Kapitel 16.7) notieren. Denn erst dann werden die Abstände zwischen Elementgrenze und Elementinhalt richtig sichtbar. Besonders sinnvoll sind die hier beschriebenen Eigenschaften daher auch für HTML-Tabellenelemente, sofern die Tabelle sichtbare Gitternetzlinien hat.



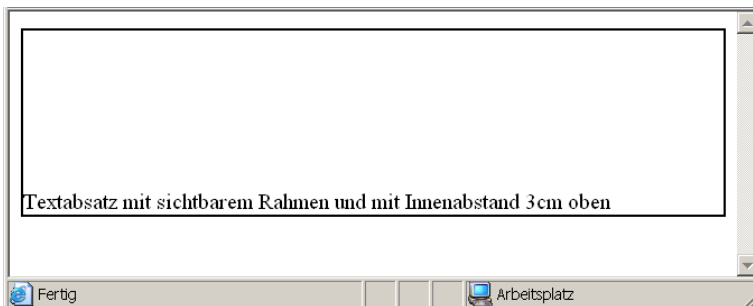
### ***padding-top (Innenabstand oben)***

Mit `padding-top`: können Sie den Zwischenraum zwischen Elementinhalt und oberer Elementgrenze (sichtbar bei Rahmen, Farben usw.) bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>padding-top</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p style="padding-top:3cm; border:thin solid black;">Textabsatz mit sichtbarem
Rahmen und
mit Innenabstand 3cm oben</p>

</body></html>
```



16.21: Ein Absatz mit 3 cm Abstand zwischen Inhalt und oberem Rand.



### ***padding-bottom (Innenabstand unten)***

Mit `padding-bottom`: können Sie den Zwischenraum zwischen Elementinhalt und unterer Elementgrenze (sichtbar bei Rahmen, Farben usw.) bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>padding-bottom</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<div style="padding-bottom:2cm; border:thin solid red;">div-Bereich mit sichtbarem
roten Rahmen und Innenabstand 2cm unten. Das ist in diesem Fall der Abstand
zwischen dem unteren Rahmen des Textabsatzes und dem unteren Rahmen des div-
Bereichs.
<p style="padding-bottom:3cm; border:thin solid black;">Textabsatz mit sichtbarem
Rahmen und mit Innenabstand 3cm unten</p>
</div>

</body></html>
```




**padding-left (Innenabstand links)**

Mit padding-left: können Sie den Zwischenraum zwischen Elementinhalt und linker Elementgrenze (sichtbar bei Rahmen, Farben usw.) bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>padding-left</title>
<style type="text/css">
#p1 { padding-left: 100px; background-color:#FFFF99; font-size:14pt; }
#p2 { padding-left: 200px; background-color:#FFFF99; font-size:14pt; }
#p3 { padding-left: 300px; background-color:#FFFF99; font-size:14pt; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<p id="p1">der erste Absatz p1</p>
<p id="p2">der zweite Absatz p2</p>
<p id="p3">der dritte Absatz p3</p>

</body></html>
```




**padding-right (Innenabstand rechts)**

Mit padding-right: können Sie den Zwischenraum zwischen Elementinhalt und rechter Elementgrenze (sichtbar bei Rahmen, Farben usw.) bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>padding-right</title>
<style type="text/css">
#d1 { padding-right: 50px; background-color:#FFFF99; font-size:14pt; }
#d2 { padding-right: 100px; background-color:#99FFFF; font-size:14pt; }
#d3 { padding-right: 150px; background-color:#FF99FF; font-size:14pt; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<div id="d1">der erste Bereich d1
<div id="d2">der zweite Bereich d2
<div id="d3">der dritte Bereich d3
</div></div></div>

</body></html>
```



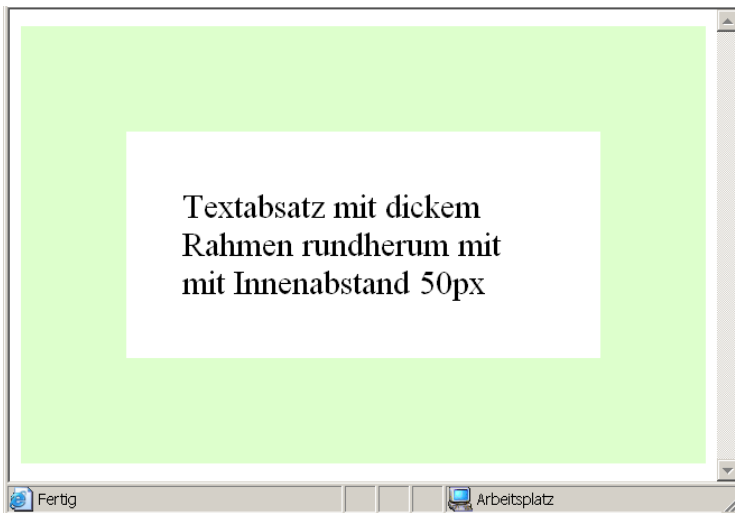

**padding (Innenabstand allgemein)**

Diese Angabe ist eine Zusammenfassung der folgenden fakultativen Einzelangaben: padding-top, padding-bottom, padding-left, padding-right. Ein Beispiel:

```
<html><head><title>padding</title>
</head><body bgcolor="#FFFFFF" text="#000000">
```



```
<p style="border:2cm solid #DDFFCC; padding:50px; font-size:18pt">Textabsatz mit  
dickem Rahmen rundherum mit mit Innenabstand 50px</p>  
  
</body></html>
```



16.22: padding: fasst verschiedene Angaben zum Innenabstand zusammen.

Mit `padding`: bestimmen Sie einen einheitlichen Innenabstand zwischen Elementinhalt und den vier Elementgrenzen oben, links, unten und rechts. Erlaubt ist eine numerische Angabe.

## 16.6 Rahmen

### *Allgemeines zu Rahmen*

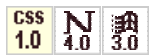
Sinnvoll sind die hier beschriebenen Angaben besonders für HTML-Elemente, die einen eigenen Absatz erzeugen bzw. einen Block bilden, also etwa für `h[1-6]`, `p`, `blockquote`, `address` oder `pre`. Blockelemente sind aber auch `div`, `table`, `tr`, `th` und `td`. So ist es beispielsweise möglich, in HTML eine blinde Tabelle (Kapitel 8.6) zu notieren, in der aber doch einzelne Zellen einen Rahmen nach Wunsch erhalten – nämlich mit Hilfe von CSS-Rahmeneigenschaften. Auch auf `body` lassen sich die hier aufgelisteten Eigenschaften anwenden – in diesem Fall wird um den gesamten sichtbaren Körper einer HTML-Datei ein Rahmen gezeichnet.

Wenn Sie bei Rahmendefinitionen nichts weiter angeben, kann es sein, dass die Inhalte von Elementen sehr nah an dem sie umgebenden Rahmen kleben. Zu diesem Zweck können Sie Innenabstände (Kapitel 16.5) definieren. Dadurch schaffen Sie einen Abstand zwischen dem Rahmen und seinem Inhalt.

In der Regel werden Sie mehrere Angaben zu Rahmen kombinieren. So ist es sinnvoll, nicht nur eine Rahmendicke anzugeben, sondern auch die gewünschte Rahmenfarbe. Auch verschiedene

Rahmentypen stehen zur Auswahl. Alle Rahmeneigenschaften können Sie wahlweise auf alle vier Seiten eines Elements anwenden, aber auch nur auf eine oder mehrere bestimmte Seiten.

**Beachten Sie:** Bei Rahmen macht sich das Blockelement-Modell von Netscape 4.x sehr negativ bemerkbar. Dieser Browser dehnt Blockelemente in der Breite nur auf die Breite seines Inhalts, sofern dieser weniger Raum einnimmt als die zur Verfügung stehende Gesamtbreite. Nur mit unschöner Trickserei lässt sich Netscape 4.x dazu überlisten, ein Blockelement auf die durch das Anzeigefenster, den Dokumentrand oder das Elternelement zur Verfügung gestellte Breite auszuweiten. Eine einheitlichere Darstellung erreichen Sie mit Angaben zur Breite von Elementen arbeiten, z.B. mit Angaben wie `width:100%` oder `width:500px`. Ein anderer, noch effektiverer Trick, der den 4er-Netscape das Element tatsächlich auf die volle zur Verfügung stehende Breite ziehen lässt, ist die Angabe `margin:0px`.



### ***boder[-top, -left, -right, -bottom]-width (Rahmendicke)***

Sie können für ein Element die Rahmendicke bestimmen. Sinnvollerweise sollten Sie diese Angabe zumindest mit einer zum Rahmentyp kombinieren, gegebenenfalls aber auch mit einer zur Rahmenfarbe. Ein Beispiel:

```
<html><head><title>border-width</title>
<style type="text/css">
#links {
    border-left-width:1cm;
    border-left-style:solid;
    border-color:red;
    padding-left:1cm;
    text-align:justify; }
#linksrechts {
    border-left-width:1cm;
    border-left-style:solid;
    border-left-color:red;
    padding-left:1cm;
    border-right-width:1cm;
    border-right-style:solid;
    border-right-color:green;
    padding-right:1cm;
    text-align:justify; }
#rundrum {
    border-width:1px;
    border-style:solid;
    border-color:blue;
    padding:1cm;
    text-align:justify; }
}
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

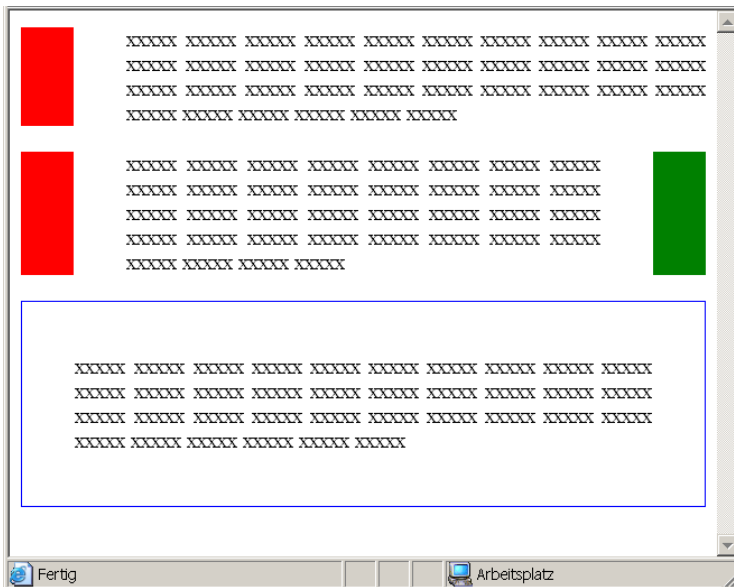
<p id="links">xxxxx xxxxx xxxxx usw.</p>
```

```

<p id="linksrechts">xxxxx xxxxx xxxxx usw.</p>
<p id="rundrum">xxxxx xxxxx xxxxx usw.</p>

</body></html>

```



16.23: Verschiedene Rahmenstärken auf einzelnen Seiten von Absätzen.

Mit `border-width`: können Sie die Dicke des Rahmens um ein Element bestimmen. Erlaubt ist eine numerische Angabe für die Rahmendicke oder einer der folgenden Werte: `thin` = dünn, `medium` = mittelstark, `thick` = dick.

Um für einzelne Seiten des Elements eine unterschiedliche Rahmendicke zu bestimmen, gibt es zwei Möglichkeiten. Die eine ist, bei `border-width` mehrere Angaben zu notieren und diese durch Leerzeichen zu trennen, zum Beispiel: `border-width:5px 5px`; Dabei gelten folgende Regeln:

- Zwei Angaben: Die erste Angabe bedeutet die Rahmendicke für oben und unten, die zweite Angabe die Rahmendicke für rechts und links.
- Drei Angaben: Die erste Angabe bedeutet die Rahmendicke für oben, die zweite die Rahmendicke für rechts und links und die dritte die Rahmendicke für unten.
- Vier Angaben: Die erste Angabe bedeutet die Rahmendicke für oben, die zweite die Rahmendicke für rechts, die dritte die Rahmendicke für unten und die vierte die Rahmendicke für links.

Die zweite Möglichkeit besteht darin, die Untereigenschaften von `border-width` zu verwenden:

- `border-top-width` definiert die Rahmendicke für oben,
- `border-left-width` die Rahmendicke für links,

- `border-right-width` die Rahmendicke für rechts,
- `border-bottom-width` die Rahmendicke für unten.

**Beachten Sie:** Der MS Internet Explorer interpretiert die Angabe `border-width` nur, wenn außerdem der Rahmentyp (`border-style`) angegeben wird, also etwa:

```
<p style="border-style:solid; border-width:thin">
```



### ***border[-top, -left, -right, -bottom]-color (Rahmenfarbe)***

Sie können für ein Element die Rahmenfarbe bestimmen. Die Angabe dazu sollten Sie mit einer zum Rahmentyp und/oder einer zur Rahmendicke kombinieren. Ein Beispiel:

```
<html><head><title>border-color</title>
</head><body bgcolor="#000000" text="#E0E0E0">

<p style="border-color:#FFFF00; border-width:2px; border-style:solid; padding:4px">
xxxxx xxxxx xxxxx usw.</p>

<p style="border-top-color:#FF00FF; border-left-color:#00FF00;
border-right-color:#3333FF; border-bottom-color:#FFFF00; border-width:4px;
border-style:solid; padding:8px">
xxxxx xxxxx xxxxx usw.</p>

<p style="border-color:cyan yellow; border-width:8px; border-style:solid;
padding:16px">
xxxxx xxxxx xxxxx usw.</p>

</body></html>
```

Mit `border-color`: können Sie die Rahmenfarbe bestimmen. Erlaubt ist eine Farbangabe oder der Wert `transparent` (durchsichtiger Rahmen, interessant bei sich überlappenden Elementen, wie z.B. durch Positionierung wie in Kapitel 16.11 beschrieben).

Um für einzelne Seiten des Elements eine unterschiedliche Rahmenfarbe zu bestimmen, gibt es zwei Möglichkeiten. Die eine ist, bei `border-color` mehrere Angaben zu notieren und diese durch Leerzeichen zu trennen, zum Beispiel: `border-color:#FFFF00 red`; Dabei gelten folgende Regeln:

- Zwei Angaben: Die erste Angabe bedeutet die Rahmenfarbe für oben und unten, die zweite Angabe die Rahmenfarbe für rechts und links.
- Drei Angaben: Die erste Angabe bedeutet die Rahmenfarbe für oben, die zweite die Rahmenfarbe für rechts und links und die dritte die Rahmenfarbe für unten.
- Vier Angaben: Die erste Angabe bedeutet die Rahmenfarbe für oben, die zweite die Rahmenfarbe für rechts, die dritte die Rahmenfarbe für unten und die vierte die Rahmenfarbe für links.

Die zweite Möglichkeit besteht darin, die Untereigenschaften von `border-color` zu verwenden:

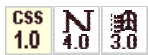
- `border-top-color` definiert die Rahmenfarbe für oben,
- `border-left-color` die Rahmenfarbe für links,

- `border-right-color` die Rahmenfarbe für rechts und
- `border-bottom-color` die Rahmenfarbe für unten.

Die Untereigenschaften von `border-color` stehen im Gegensatz zu den Untereigenschaften von `border-width` jedoch erst seit CSS 2.0 zur Verfügung und werden von älteren Browsern ignoriert.

**Beachten Sie:** Der MS Internet Explorer interpretiert die Angabe `border-width` nur, wenn außerdem der Rahmentyp (`border-style`) angegeben wird, also etwa: `<p style="border-style:solid; border-color:red">`.

Netscape 4.x hat mit Angaben zur Rahmenfarbe zum Teil erhebliche Probleme und interpretiert teilweise augenscheinlich unsinniges Zeug. Das obige Beispiel etwa lässt ihn böse straucheln.



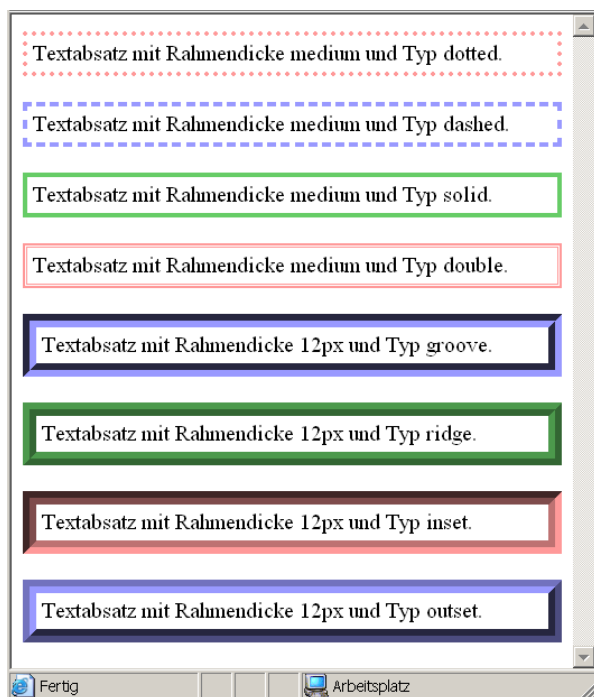
### ***border[-top, -left, -right, -bottom]-style (Rahmentyp)***

Mit dem Rahmentyp können Sie verschiedene Schmuckrahmen definieren. Damit die Typen zur Geltung kommen, empfiehlt sich die Kombination mit Angaben zur Rahmendicke und/oder zur Rahmenfarbe. Ein Beispiel:

```
<html><head><title>border-style</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p style="border-width:medium; border-color:#FF9999;
  border-style:dotted; padding:5px;">
Textabsatz mit Rahmendicke medium und Typ dotted.</p>
<p style="border-width:medium; border-color:#9999FF;
  border-style:dashed; padding:5px;">
Textabsatz mit Rahmendicke medium und Typ dashed.</p>
<p style="border-width:medium; border-color:#66CC66;
  border-style:solid; padding:5px;">
Textabsatz mit Rahmendicke medium und Typ solid.</p>
<p style="border-width:medium; border-color:#FF9999;
  border-style:double; padding:5px;">
Textabsatz mit Rahmendicke medium und Typ double.</p>
<p style="border-width:12px; border-color:#9999FF;
  border-style:groove; padding:5px;">
Textabsatz mit Rahmendicke 12px und Typ groove.</p>
<p style="border-width:12px; border-color:#66CC66;
  border-style:ridge; padding:5px;">
Textabsatz mit Rahmendicke 12px und Typ ridge.</p>
<p style="border-width:12px; border-color:#FF9999;
  border-style:inset; padding:5px;">
Textabsatz mit Rahmendicke 12px und Typ inset.</p>
<p style="border-width:12px; border-color:#9999FF;
  border-style:outset; padding:5px;">
Textabsatz mit Rahmendicke 12px und Typ outset.</p>

</body></html>
```



16.24: Mit CSS lässt sich eine große Vielfalt an Rahmentypen erzeugen.

Mit `border-style`: können Sie den Rahmentyp festlegen. Erlaubt ist eine der folgenden Angaben:

| CSS-Angabe          | Wirkung                                |
|---------------------|----------------------------------------|
| <code>none</code>   | kein Rahmen (bzw. unsichtbarer Rahmen) |
| <code>hidden</code> | kein Rahmen (bzw. unsichtbarer Rahmen) |
| <code>dotted</code> | gepunktet                              |
| <code>dashed</code> | gestrichelt                            |
| <code>solid</code>  | durchgezogen                           |
| <code>double</code> | doppelt durchgezogen                   |
| <code>groove</code> | 3-D-Effekt                             |
| <code>ridge</code>  | 3-D-Effekt                             |
| <code>inset</code>  | 3-D-Effekt                             |
| <code>outset</code> | 3-D-Effekt                             |

Um für einzelne Seiten des Elements einen unterschiedlichen Rahmentyp zu bestimmen, gibt es zwei Möglichkeiten. Die eine ist, bei `border-style` mehrere Angaben zu notieren und diese durch Leerzeichen zu trennen, zum Beispiel: `border-style:solid double`; dabei gelten folgende Regeln:

- Zwei Angaben: Die erste Angabe bedeutet den Rahmentyp für oben und unten, die zweite Angabe den Rahmentyp für rechts und links.
- Drei Angaben: Die erste Angabe bedeutet den Rahmentyp für oben, die zweite den Rahmentyp für rechts und links und die dritte den Rahmentyp für unten.
- Vier Angaben: Die erste Angabe bedeutet den Rahmentyp für oben, die zweite den Rahmentyp für rechts, die dritte den Rahmentyp für unten und die vierte den Rahmentyp für links.

Die zweite Möglichkeit besteht darin, die Untereigenschaften von `border-style` zu verwenden:

- `border-top-style` definiert den Rahmentyp für oben,
- `border-left-style` den Rahmentyp für links,
- `border-right-style` den Rahmentyp für rechts und
- `border-bottom-style` den Rahmentyp für unten.

Die Untereigenschaften von `border-style` stehen im Gegensatz zu den Untereigenschaften von `border-width` jedoch erst seit CSS 2.0 zur Verfügung und werden von älteren Browsern ignoriert.

**Beachten Sie:** Die Angabe `hidden` ist vor allem für benachbarte Tabellenzellen gedacht. Während die Angabe `none` nur den Rahmen des Elements unterdrückt, für das sie gilt, bewirkt `hidden`, dass auch Rahmen von Nachbarelementen unterdrückt werden, sofern diese Rahmen besitzen. Voraussetzung dafür ist allerdings, dass für das zugehörige `table`-Element die folgende CSS-Sondereigenschaft notiert ist:

```
border-collapse: collapse
```

Dies ist eine neuere CSS-Eigenschaft, die Rahmenregeln für Tabellen erlaubt. Beim Wert `collapse` fallen alle Rahmen der Zellen sauber übereinander, d.h., eine Zelle mit rechtem Rand von 1px und eine Zelle rechts davon mit einem linken Rand von 1px bilden einen gemeinsamen Rahmen von 1px. Der gegenteilige mögliche Wert, der allerdings auch der Voreinstellung entspricht, lautet `separate`. Damit »stoßen« die Rahmen benachbarter Zellen nur aneinander, überlappen sich aber nicht. Im Beispiel der zwei erwähnten Zellen würde also optisch ein Zwischenrahmen von 2 px Dicke entstehen.

Die `border-collapse`-Eigenschaft wird bislang nur vom Internet Explorer 5.x und vom Opera-Browser interpretiert. Netscape 4.x interpretiert `border-style` nur, wenn außerdem eine Angabe zur Rahmendicke notiert wird, also etwa:

```
<p style="border-style:solid; border-width:thin">
```

Beide Browser interpretieren die Angaben zu `border-style` nicht vollständig. Einige der Effekte (z.B. `groove`) kommen nur zustande, wenn Sie eine Farbe angeben, die sich von schwarz unterscheidet.



### ***border[-top, -left, -right, -bottom] (Rahmen)***

Diese Angabe ist eine Zusammenfassung der folgenden möglichen Einzelangaben: `border-width`, `border-style`, `border-color`.

Ein Beispiel:

```
<html><head><title>border</title>
<style type="text/css">
h1 { border-bottom:4px double blue; padding-bottom:4px; margin:0px; }
h2 { border-top:1px solid blue; margin:0px; }
p.info { border:1px solid grey; background-color:#E0E0E0; padding:4px; margin:0px;
}
body { font-family:Tahoma,Helvetica,sans-serif; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<h1>Überschrift erster Ordnung</h1>

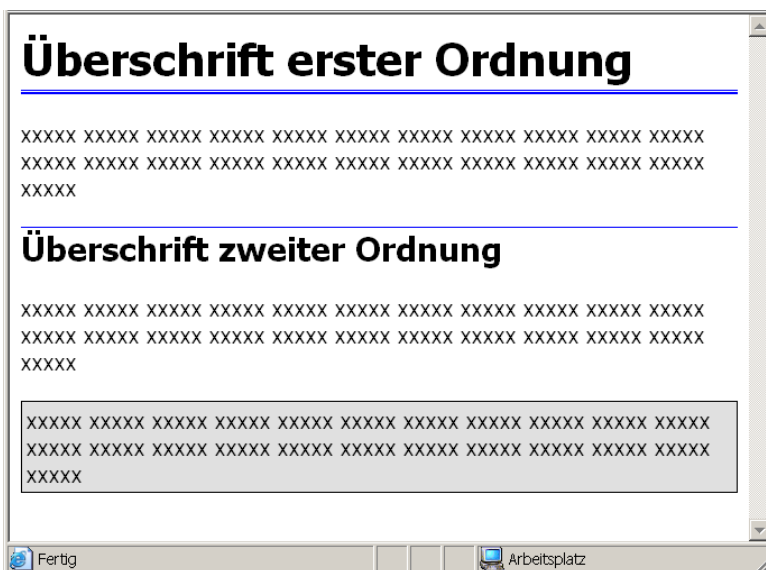
<p>xxxxx xxxxx xxxxx usw.</p>

<h2>Überschrift zweiter Ordnung</h2>

<p>xxxxx xxxxx xxxxx usw.</p>

<p class="info">xxxxx xxxxx xxxxx usw.</p>

</body></html>
```



16.25: Über eine Kombination verschiedener Rahmenangaben entstehen zum Beispiel doppelt unterstrichene Überschriften.

Mit `border`: können Sie das Aussehen eines Rahmens um ein Element bestimmen. Notieren Sie Angaben zu Linienstärke, Rahmentyp und Farbe mit Leerzeichen dazwischen, so wie in den



Beispielen oben. Die Reihenfolge der Einzelangaben ist egal. Daneben können Sie zusammenhängende Angaben für Rahmendicke, Rahmenfarbe und Rahmentyp auch für einzelne Seiten des Elements machen. Dazu sind die Untereigenschaften von `border` gedacht:

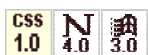
- `border-top` definiert eine zusammenhängende Angabe für oben,
- `border-left` für links,
- `border-right` für rechts und
- `border-bottom` für unten.

## 16.7 Hintergrundfarben und -bilder

### Allgemeines zu Hintergrundfarben und -bildern

Sinnvoll sind die hier beschriebenen CSS-Eigenschaften vor allem für HTML-Elemente, die einen eigenen Absatz erzeugen bzw. einen Block bilden, also etwa für `h[1-6]`, `p`, `blockquote`, `address` oder `pre`. Blockelemente sind aber auch `div`, `table`, `tr`, `th` und `td`. Auch auf `<body>` lassen sich die hier aufgelisteten Stylesheet-Angaben anwenden – in diesem Fall beziehen sich die Angaben auf den gesamten sichtbaren Körper der HTML-Datei. Es ist aber ebenso möglich, einzelne Textpassagen, die durch ein Inline-Element wie `span`, `b`, `strong` usw. markiert sind, mit einem eigenen Hintergrund zu belegen.

Elementspezifische Hintergrundangaben heben sich von eventuellen HTML-Angaben zur Hintergrundfarbe der HTML-Datei oder zu einer dateiweiten Hintergrundgrafik ab (Kapitel 4.2). Sehr schöne Effekte können Sie erzielen, indem Sie Angaben zu Hintergrundfarben bzw. Hintergrundbildern einzelner Elemente mit Angaben zu Rahmen (Kapitel 16.6) und Innenabstand (Kapitel 16.5) kombinieren.



### **background-color (Hintergrundfarbe)**

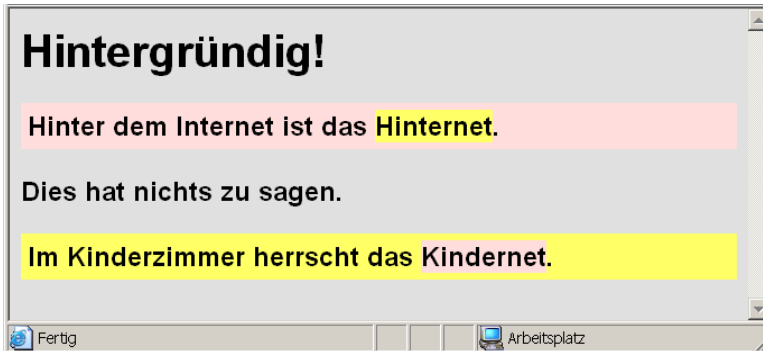
Sie können mit `background-color:` für ein HTML-Element eine eigene Hintergrundfarbe definieren. Ein Beispiel:

```
<html><head><title>background-color</title>
<style type="text/css">
body { background-color:#E0E0E0; font-weight:bold; font-family:Arial; font-
size:120%; }
.rosabox { background-color:#FFDDDD; padding:6px; margin:0px; }
.rosa { background-color:#FFDDDD; }
.gelbbox { background-color:#FFFF66; padding:6px; margin:0px; }
.gelb { background-color:#FFFF66; }
</style>
</head><body>

<h1>Hintergrund!</h1>
<p class="rosabox">Hinter dem Internet ist das <span
class="gelb">Hinternet</span>.</p>
```

```
<p>Dies hat nichts zu sagen.</p>
<p class="gelbbox">Im Kinderzimmer herrscht das <span
class="rosa">Kindernet</span>.</p>

</body></html>
```



16.26: CSS erlaubt eigene Hintergrundfarben für praktisch jedes Element.

**Beachten Sie:** Netscape 4.x bildet bei mehrzeiligen Textabsätzen keinen echten Farbblock, sondern dehnt die Hintergrundfarbe am Zeilenende nur bis zum zufälligen Ende des Textes aus. Um dies zu verhindern, können Sie zusätzlich die Angaben `border:none;` und `margin:0px;` notieren (in den meisten Fällen kommen Sie auch ohne das `margin:0px;` aus). Wenn Ihnen der Abstand zwischen Textgrenzen und dem Farbblock, der durch die Hintergrundfarbe entsteht, zu klein ist, können Sie wie im obigen Beispiel gezeigt mit `padding` einen Innenabstand definieren.



### **background-image (Hintergrundbild)**

Sie können für einzelne HTML-Elemente ein eigenes Hintergrundbild definieren. Ein Beispiel:

```
<html><head><title>background-image</title>
</head><body style="background-image:url(background1.gif)">

<div style="background-image:url(background2.gif); margin:0px; padding:10px">
<h1 align="center" style="color:#FFFFCC">Willkommen im Reich der Texturen!</h1>

<div style="background-image:url(background3.jpg); margin:30px; padding:10px">
<h2 align="center" style="color:#804000">Willkommen im Reich der Texturen!</h2>

</div></div>

</body></html>
```



16.27: Auch ein Hintergrundbild ist für Elemente einer Web-Seite kein Problem.

Mit `background-image:url([URI]):` können Sie eine Hintergrundgrafik bestimmen. Per Voreinstellung wird die Hintergrundgrafik als Wallpaper (Tapetenmuster) wiederholt, so wie bei der Angabe eines Hintergrundbildes im einleitenden HTML-Tag `<body>`. Text und referenzierte Grafiken erscheinen über dieser Hintergrundgrafik. Als Grafikdateitypen sollten Sie wie in HTML üblich GIF- oder JPG-Grafiken benutzen.

Im Beispiel wird vorausgesetzt, dass sich die Grafikdateien im gleichen Verzeichnis befinden wie die HTML-Datei. Wenn die Grafik in einem anderen Verzeichnis steht, müssen Sie den relativen oder absoluten Pfadnamen angeben. Das funktioniert genau so wie beim Einbinden von Grafiken in HTML (Kapitel 7.1).

**Beachten Sie:** Wenn das Element, für das die Hintergrundgrafik definiert wird, in der Höhe oder Breite kleiner ist als die Hintergrundgrafik, wird die Hintergrundgrafik in der Darstellung an den Grenzen des Elements abgeschnitten.



### ***background-repeat (Wiederholungseffekt)***

Normalerweise wird eine Hintergrundgrafik im Wallpaper-Effekt über den gesamten zur Verfügung stehenden Raum wiederholt. Sie können jedoch ein anderes Verhalten erzwingen. Ein Beispiel:

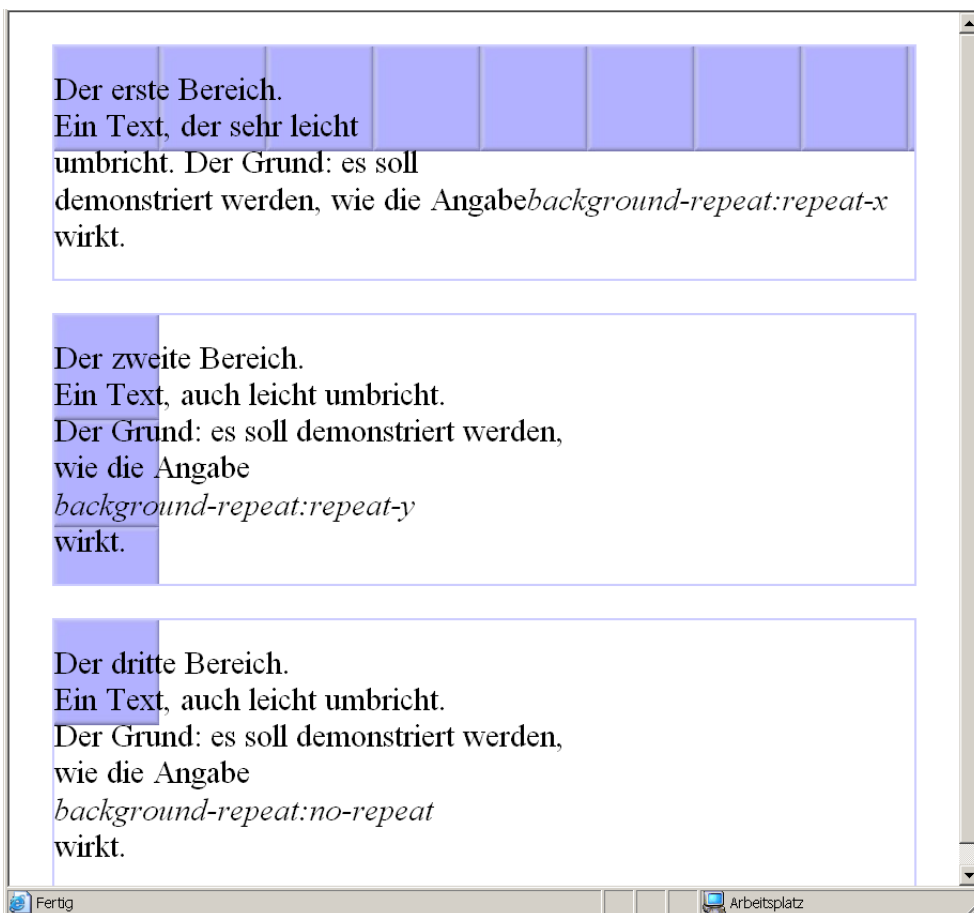
```
<html><head><title>background-repeat</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<div style="background-image:url(background4.jpg); background-repeat:repeat-x;
margin:30px; border:thin solid #CCCCFF; font-size:150%">
<p>Der erste Bereich.<br>Ein Text, der sehr leicht<br>umbricht...usw.</p>
</div>
```

```

<div style="background-image:url(background4.jpg); background-repeat:repeat-y;
margin:30px; border:thin solid #CCCCFF; font-size:150%">
<p>Der zweite Bereich.<br>Ein Text, auch leicht umbricht...usw.</p>
</div>
<div style="background-image:url(background4.jpg); background-repeat:no-repeat;
margin:30px; border:thin solid #CCCCFF; font-size:150%">
<p>Der dritte Bereich.<br>Ein Text, auch leicht umbricht...usw.</p>
</div>
</body></html>

```



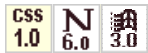
16.28: Das Hintergrundbild wird nur waagrecht, senkrecht oder gar nicht gekachelt.

Mit `background-repeat`: können Sie das Wiederholungsverhalten einer Hintergrundgrafik, die Sie mit `background-image` einbinden, kontrollieren. Erlaubt ist eine der folgenden Angaben:

- `repeat` = wiederholen (Voreinstellung)
- `repeat-x` = nur »eine Zeile lang« waagrecht wiederholen

- `repeat-y` = nur »eine Spalte lang« senkrecht wiederholen
- `no-repeat` = nicht wiederholen, nur als Einzelbild anzeigen

**Beachten Sie:** Netscape 4.x interpretiert diese Eigenschaft zwar, hat aber massive Schwierigkeiten bei der Darstellung von Elementinhalten.



### ***background-attachment (Wasserzeichen-Effekt)***

Bei längeren Elementen wandert ein Hintergrundbild beim Scrollen im Text optisch mit. Sie können jedoch erzwingen, dass das Hintergrund stehen bleibt (Wasserzeichen-Effekt). Ein Beispiel:

```
<html><head><title>background-attachment</title>
<style type="text/css">
body {
  background-image:url(background5.gif);
  background-repeat:no-repeat;
  background-attachment:fixed; padding:0px; }
div.mybody {
  margin-left:100px; margin-top:20px; margin-right:20px;
  margin-bottom:20px; }
</style>
</head><body>

<div class="mybody">
<h1>viel Text<br>viel Text zum Scrollen!<!--usw.--></h1>
</div>

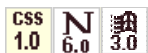
</body></html>
```

Mit `background-attachment`: können Sie das Scroll-Verhalten einer Hintergrundgrafik, die Sie mit `background-image` einbinden, kontrollieren. Erlaubt ist eine der folgenden Angaben:

- `scroll` = mitscrollen (Voreinstellung)
- `fixed` = Hintergrundbild bleibt stehen

Das obige Beispiel demonstriert einen interessanten Anwendungsfall aus der Kombination der Eigenschaften `background-attachment` und `background-repeat` sowie Angaben zum Außenrand und Innenabstand (Kapitel 16.4 und 16.5). Eine Hintergrundgrafik wird dem `body`-Element zugewiesen und wiederholungslos in der obersten linken Ecke angezeigt. Der gesamte Inhalt des Dokuments kommt in einen `div`-Bereich mit der Klasse `mybody`, der seinen linken Außenrand so definiert hat, dass der Inhalt des Dokuments beim Scrollen an der Grafik vorbeifließt, die immer angezeigt bleibt. So könnte manches Frameset vermieden werden!

**Beachten Sie:** Netscape 4.x interpretiert diese Angabe nicht.



### ***background-position (Hintergrundposition)***

Diese Angabe ist vor allem dann interessant, wenn Sie mit einer entsprechenden Angabe zu `background-repeat` erzwingen, dass eine Hintergrundgrafik nur einmal angezeigt wird. Für

diesen Fall können Sie mit der Hintergrundposition festlegen, wo genau die Hintergrundgrafik innerhalb des HTML-Elements platziert werden soll. Ein Beispiel:

```
<html><head><title>background-position</title>
<style type="text/css">
#Tanz {
  background-image:url(background6.gif);
  background-repeat:no-repeat;
  background-position:30px 20px;
}
</style>
</head><body>

<div id="Tanz">
<h1>Tanz der Bilder!</h1>

</div>

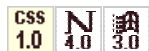
</body></html>
```

Mit `background-position`: können Sie festlegen, wo die linke obere Ecke der Hintergrundgrafik sein soll. Bezugspunkt ist das HTML-Element, für das die Hintergrundgrafik definiert wird. Erlaubt sind numerische Angaben und folgende Angaben:

| CSS-Angabe | Wirkung                 |
|------------|-------------------------|
| top        | vertikal obenbündig     |
| center     | horizontal zentriert    |
| middle     | vertikal mittig         |
| bottom     | vertikal untenbündig    |
| left       | horizontal linksbündig  |
| right      | horizontal rechtsbündig |

Im obigen Beispiel enthält das `div`-Element eine GIF-Grafik mit transparentem Hintergrund. Diese wird im Beispiel über der Hintergrundgrafik angezeigt. Es sieht aus wie eine Grafik, sind aber zwei verschiedene.

**Beachten Sie:** Netscape 4.x interpretiert diese Angabe nicht.



### **background (Hintergrund allgemein)**

Diese Angabe ist eine Zusammenfassung der folgenden möglichen Einzelangaben: `background-image` (Hintergrundbild), `background-repeat` (Wiederholungseffekt), `background-attachment` (Wasserzeichen-Effekt), `background-position` (Hintergrundposition). Ein Beispiel:

```
<html><head><title>background</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="background:url(background1.gif) repeat-x;">
```

```
<h1 style="font-family:Arial,sans-serif; color:maroon; font-size:60pt">NEWS</h1>
<div style="margin-top:45px; font-size:14pt">
<p>Hier folgen die News...</p>
</div></div>
</body></html>
```



16.29: Eine Kombination der Effekte zum Hintergrundbild erlaubt auffällige Designs.

Mit `background`: können Sie Angaben für eine Hintergrundgrafik zusammenfassen. Notieren Sie Angaben zum Hintergrundbild, zu Wiederholungs- und Wasserzeichen-Effekt sowie zur Position mit Leerzeichen dazwischen, so wie im Beispiel oben. Die Reihenfolge der Einzelangaben ist egal. Es ist nicht erforderlich, zu allen Angaben etwas zu notieren.

**Beachten Sie:** Netscape 4.x interpretiert diese Angabe zwar, aber was er aus dem obigen Beispiel macht, spottet leider jeder Beschreibung.

## 16.8 Listenformatierung

### Allgemeines zur Listenformatierung

Die Stylesheet-Angaben für Listen beziehen sich speziell auf die HTML-Elemente `ul` (Aufzählungslisten, Kapitel 5.4.1) und `ol` (nummerierte Listen, Kapitel 5.4.2). Sie können mit Hilfe von Stylesheet-Angaben die gleichen Formatierungen erzwingen, wie es in den einleitenden HTML-Tags `<ul>` und `<ol>` mit Hilfe von zusätzlichen Attributen möglich ist. Die entsprechenden Attribute sind in HTML 4.0 jedoch alle als *deprecated* ausgewiesen, also als künftig unerwünscht. Stattdessen sollen die hier beschriebenen CSS-Eigenschaften verwendet werden, um solche Listen zu formatieren. Neben den Möglichkeiten, die HTML bietet, stehen auch erweiterte Möglichkeiten zum Formatieren von Listen zur Verfügung, etwa die Möglichkeit, Listen aus- statt einzurücken oder eine eigene Grafik als Aufzählungssymbol zu definieren.

### 16.8.1 list-style-type (Darstellungstyp)

Für Aufzählungslisten können Sie das Erscheinungsbild des Bullet-Zeichens bestimmen. Für nummerierte Listen können Sie die Darstellung der Durchnummerierung beeinflussen.

Ein Beispiel:

```
<html><head><title>list-style-type</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h4>Aufzählungsliste &lt;ul> mit Darstellungstyp <i>disc</i></h4>
<ul style="list-style-type:disc">
<li>Probieren geht &uuml;ber Studieren</li>
<!--usw.-->
</ul>
<h4>Aufzählungsliste &lt;ul> mit Darstellungstyp <i>circle</i></h4>
<ul style="list-style-type:circle">
<li>Probieren geht &uuml;ber Studieren</li>
<!--usw.-->
</ul>
<!--usw.-->

<h4>Nummerierte Liste &lt;ol> mit Darstellungstyp <i>decimal</i></h4>
<ol style="list-style-type:decimal">
<li>Tagesordnung</li>
<!--usw.-->
</ol>
<h4>Nummerierte Liste &lt;ol> mit Darstellungstyp <i>lower-roman</i></h4>
<ol style="list-style-type:lower-roman">
<li>Tagesordnung</li>
<!--usw.-->
</ol>

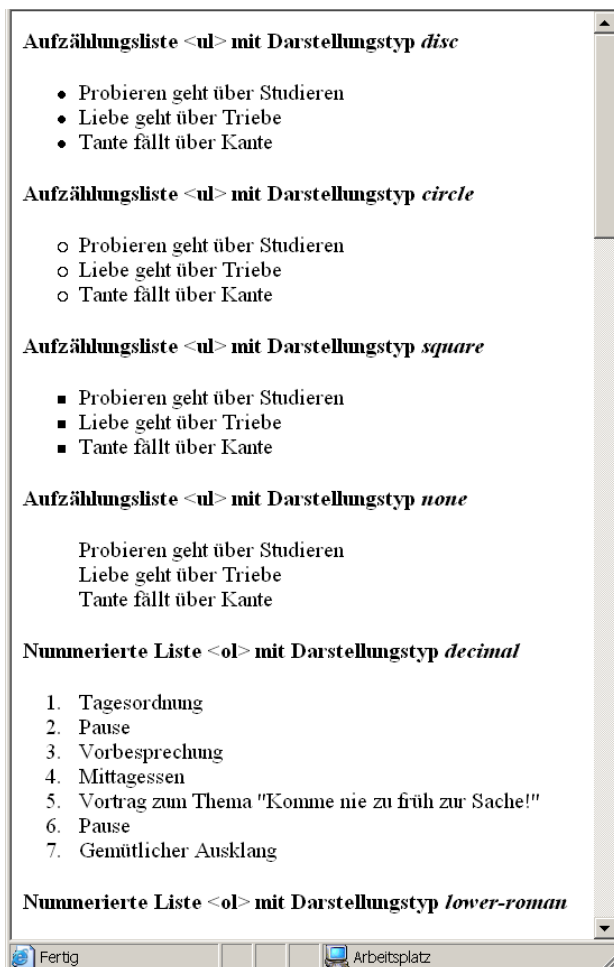
<!--usw.-->
</body></html>
```

Mit `list-style-type`: können Sie das Aussehen von Listenzeichen kontrollieren. Erlaubt ist eine der folgenden Angaben.

| CSS-Angabe               | Wirkung                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------|
| <code>decimal</code>     | für <code>ol</code> -Listen: Nummerierung 1.,2.,3.,4. usw.                                         |
| <code>lower-roman</code> | für <code>ol</code> -Listen: Nummerierung i.,ii.,iii.,iv. usw.                                     |
| <code>upper-roman</code> | für <code>ol</code> -Listen: Nummerierung I.,II.,III.,IV. usw.                                     |
| <code>lower-alpha</code> | für <code>ol</code> -Listen: Nummerierung a.,b.,c.,d. usw.                                         |
| <code>upper-alpha</code> | für <code>ol</code> -Listen: Nummerierung A.,B.,C.,D. usw.                                         |
| <code>lower-latin</code> | für <code>ol</code> -Listen: Nummerierung a.,b.,c.,d. usw. inklusive erweitertem Latin-Zeichensatz |



| CSS-Angabe  | Wirkung                                                                              |
|-------------|--------------------------------------------------------------------------------------|
| upper-latin | für ol-Listen: Nummerierung A.,B.,C.,D. usw. inklusive erweitertem Latin-Zeichensatz |
| disc        | für ul-Listen: Dateisymbol als Bullet-Zeichen                                        |
| circle      | für ul-Listen: rundes Bullet-Zeichen                                                 |
| square      | für ul-Listen: rechteckiges Bullet-Zeichen                                           |
| none        | kein Bullet-Zeichen, keine Nummerierung                                              |

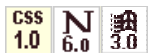


16.30: Verschiedene Arten von Nummerierungen und Bullet-Zeichen für Listen.



Die folgenden, neueren CSS-Angaben sind ebenfalls möglich, werden bislang jedoch nur von Netscape ab Version 6.1 interpretiert.

| CSS-Angabe           | Wirkung                                                                           |
|----------------------|-----------------------------------------------------------------------------------|
| lower-greek          | für ol-Listen: Nummerierung mit den griechischen Buchstaben                       |
| hebrew               | für ol-Listen: Nummerierung mit den hebräischen Buchstaben                        |
| decimal-leading-zero | für ol-Listen: Nummerierung mit führender 0: 01.,02.,03.,04., ... 98.,99.<br>usw. |
| cjk-ideographic      | für ol-Listen: Nummerierung mit ideographischen Zeichen                           |
| hiragana             | für ol-Listen: Nummerierung in Japanisch (a, i, u, e, o, ka, ki, ... )            |
| katakana             | für ol-Listen: Nummerierung in Japanisch (A, I, U, E, O, KA, KI, ... )            |
| hiragana-iroha       | für ol-Listen: Nummerierung in Japanisch (i, ro, ha, ni, ho, he, to, ... )        |
| katakana-iroha       | für ol-Listen: Nummerierung in Japanisch (I, RO, HA, NI, HO, HE, TO, ... )        |



### list-style-position (Listeneinrückung)

Sie können bestimmen, wie sich Bullet-Zeichen oder Nummerierung mit dem Inhalt eines Listeneintrags in Bezug auf Einrückung verhalten. Ein Beispiel:

```
<html><head><title>list-style-position</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h4>Nummerierte Liste &lt;ol> mit Listeneinrückung <i>inside</i></h4>
<ol style="list-style-position:inside">
<li>Tagesordnung<br>
zweizeilig</li>
<!--usw.-->
</ol>

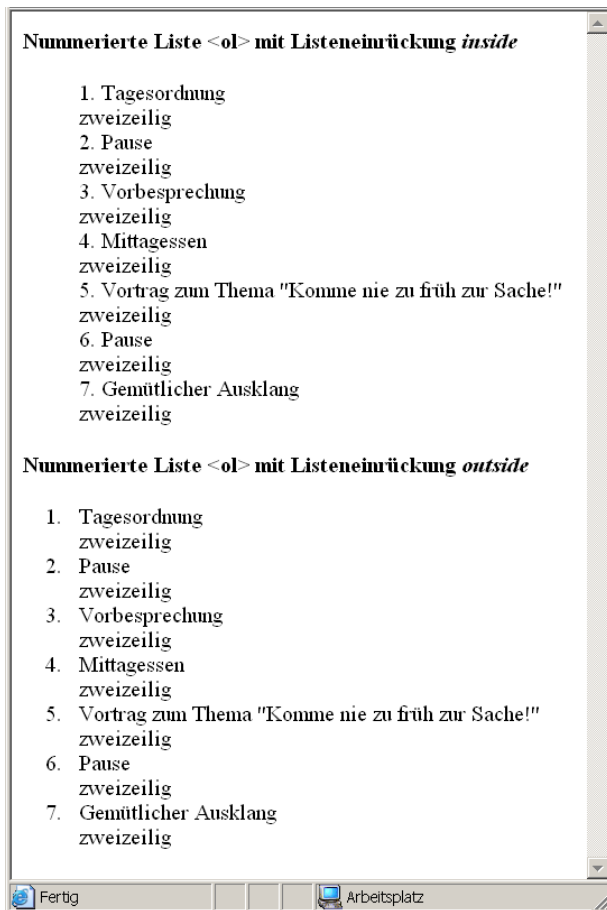
<h4>Nummerierte Liste &lt;ol> mit Listeneinrückung <i>outside</i></h4>
<ol style="list-style-position:outside">
<li>Tagesordnung<br>
zweizeilig</li>
<!--usw.-->
</ol>

</body></html>
```

Mit `list-style-position`: können Sie das Einrückungsverhalten kontrollieren. Erlaubt ist eine der folgenden Angaben.

- `inside` = eingerückt
- `outside` = ausgerückt (Voreinstellung)

**Beachten Sie:** Netscape 4.x interpretiert diese Angabe noch nicht.



16.31: Die Einrückung bei Listen lässt sich ein- oder ausschalten.

***list-style-image (eigene Bullet-Grafik)***

Mit `list-style-image:url([Dateiname])` können Sie eine Grafik für das eigene Bullet-Zeichen angeben. Als Grafikdateitypen sollten Sie wie in HTML üblich GIF- oder JPG-Grafiken benutzen. Im Beispiel wird vorausgesetzt, dass sich die Grafikdatei im gleichen Verzeichnis befindet wie die HTML-Datei. Wenn die Grafik in einem anderen Verzeichnis steht, müssen Sie den relativen oder absoluten Pfadnamen angeben. Das funktioniert genau so wie beim Einbinden von Grafiken in HTML (Kapitel 7.1.2). Ein Beispiel:

```
<html><head><title>list-style-image</title>
</head><body bgcolor="FFFFFF" text="#000000">

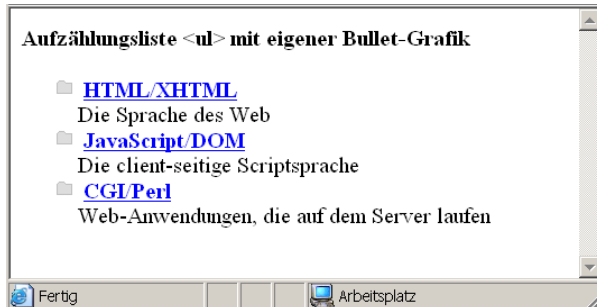
<h4>Aufz&auml;hlungsliste &lt;u> mit eigener Bullet-Grafik</h4>
<ul style="list-style-image:url(list_style_image.gif)">
<li>&#160;<a href="../../html/index.htm"><b>HTML/XHTML</b></a><br>
```

```

Die Sprache des Web</li>
<!--usw.-->
</ul>

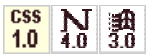
</body></html>

```



16.32: Mit CSS verwenden Sie beliebige Grafiken als Bullet-Zeichen.

**Beachten Sie:** Netscape 4.x interpretiert diese Angabe noch nicht.



### **list-style (Listendarstellung allgemein)**

Diese Angabe ist eine Zusammenfassung der folgenden möglichen Einzelangaben: list-style-type (Darstellungstyp), list-style-position (Listeneinrückung), list-style-image (Eigene Bullet-Grafik). Ein Beispiel:

```

<html><head><title>list-style</title>
</head><body bgcolor="FFFFFF" text="#000000">

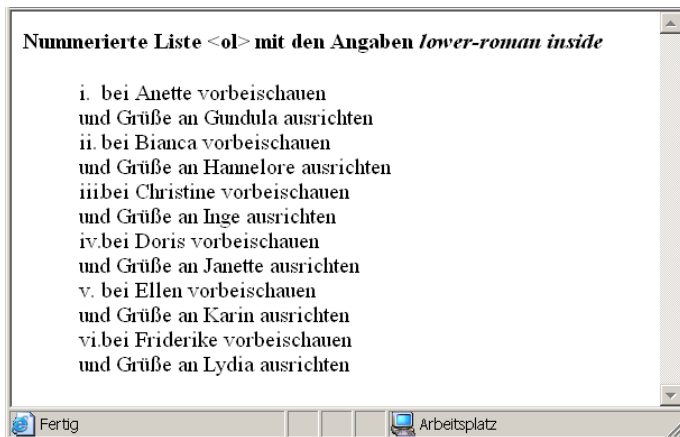
<h4>Nummerierte Liste &lt;ol> mit den Angaben <i>lower-roman inside</i></h4>
<ol style="list-style:lower-roman inside">
<li>bei Anette vorbeischauen<br>
und Gr&uuml;&szlig;e an Gundula ausrichten</li>
<!--usw.-->
</ol>

</body></html>

```

Mit list-style: können Sie zusammenfassende Angaben zum Aussehen von Aufzählungslisten oder nummerierten Listen machen. Notieren Sie Angaben zum Darstellungstyp, zur Listeneinrückung sowie zur Bullet-Grafik mit Leerzeichen dazwischen, so wie im Beispiel oben. Die Reihenfolge der Einzelangaben ist egal. Es ist nicht erforderlich, zu allen Angaben etwas zu notieren.

**Beachten Sie:** Netscape 4.x interpretiert nur einen Teil dieser Angaben.



16.33: Kombinieren Sie mehrere CSS-Angaben zu Listen mit `list-style`:

## 16.9 Tabellenformatierung

### Allgemeines zur Tabellenformatierung

Tabellen werden erst seit CSS 2.0 gesondert behandelt und haben eigene CSS-Eigenschaften. Dennoch gibt es auch eine Menge Eigenschaften von CSS 1.0, die sich auf Tabellen, Spalten und Zeilen anwenden lassen. Auf die HTML-Elemente `table`, `tr`, `colgroup`, `col`, `thead`, `tbody`, `tfoot`, `th` und `td` lassen sich Formatierungen anwenden wie:

- Schriftformatierung
- Ausrichtung und Absatzkontrolle
- Innenabstand
- Rahmen
- Hintergrundfarben und -bilder

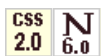
Vor allem aber bei Rahmen genügen die dafür üblichen CSS-Eigenschaften alleine nicht, um den Anforderungen an das Gitternetz einer Tabelle gerecht zu werden. Deshalb bietet CSS 2.0 die Möglichkeit an, zwei verschiedene Rahmenmodelle zu unterscheiden. Dadurch kann dem Browser mitgeteilt werden, wie er die Außenrahmen zweier angrenzender Tabellenzellen behandeln soll.

Für Angaben zur Breite und Höhe von Tabellenspalten oder -zeilen sind die CSS-Eigenschaften `width` und `height` zwar geeignet, doch ist es wirklich manchmal wünschenswert, dass vorgegebene Breiten und Höhenangaben in jedem Fall und auch mal mit Priorität vor dem Inhalt der Zellen behandelt werden. Diese Option ist nun mit CSS 2.0 für Tabellen einstellbar.

Andere tabellenspezifische Eigenschaften regeln Angaben, die auch in HTML in Form von Attributen möglich sind. Die entsprechenden Attribute sind in HTML 4.0 jedoch als *deprecated*

gekennzeichnet, also als künftig unerwünscht. Stattdessen sollen die hier beschriebenen CSS-Eigenschaften verwendet werden.

Weiterhin wichtig im Zusammenhang mit Tabellen ist seit CSS 2.0 die an anderer Stelle beschriebene CSS-Eigenschaft `display` (Kapitel 16.11). Damit ist es möglich, Tabelleneigenschaften auf andere Elemente als Tabellenelemente anzuwenden. Das mag sich zwar für HTML komisch anhören, aber CSS ist ja auch eine mögliche Style-Sprache zur Darstellung von XML-Daten. Und da es in XML keine Möglichkeit gibt, ein Element mit semantischen Eigenschaften wie »Tabellenzelle« oder »Tabellenzeile« auszuzeichnen, muss das die Style-Sprache übernehmen, die solche Daten anzeigt.



### *caption-side (Ausrichtung der Tabellenüberschrift)*

Sie können für Tabellen in HTML mit dem `caption`-Element eine Tabellenüberschrift bestimmen. Mit der hier beschriebenen Stylesheet-Angabe können Sie die Position dieser Überschrift festlegen. Beispiel:

```
<html><head><title>caption-side</title>
</head><body bgcolor="FFFFFF" text="#000000">

<table border="1">
<caption style="caption-side:bottom">Assoziationen</caption>
<tr>
<th>Berlin</th><th>Hamburg</th><th>M&uuml;nchen</th>
</tr><tr>
<td>Milj&ouml;h</td><td>Kiez</td><td>Bierdampf</td>
</tr><tr>
<td>Buletten</td><td>Frikadellen</td><td>Fleischpflanzerl</td>
</tr></table>

</body></html>
```

Mit `caption-side`: können Sie die Position der Tabellenüberschrift bestimmen. Folgende Angaben sind erlaubt:

| CSS-Angabe | Wirkung                  |
|------------|--------------------------|
| top        | oberhalb der Tabelle     |
| bottom     | unterhalb der Tabelle    |
| left       | links neben der Tabelle  |
| right      | rechts neben der Tabelle |

**Beachten Sie:** Diese Eigenschaft wird vom Internet Explorer in der Version 5.x noch nicht interpretiert.



### *table-layout (fixe/variable Breiten)*

Mit dieser Eigenschaft können Sie entscheiden, wie der Browser Breitenangaben zur Tabelle und zu Tabellenspalten behandeln soll, die Sie mit der CSS-Eigenschaft `width` angeben. Dies hat den Vorteil, dass große Tabellen schneller angezeigt werden können, da der Browser nicht mehr erst den gesamten Inhalt der Tabelle einlesen muss, um die tatsächliche Anzeigebreite einzulesen. Ein Beispiel:

```
<html><head><title>table-layout</title>
</head><body bgcolor="FFFFFF" text="#000000">

<table border="1" style="table-layout:fixed">
<colgroup span="3" style=""></colgroup>
<tr>
<td style="width:100px">
123456789012345678901234567890123456789012345678901234567890
</td>
<td style="width:200px">
123456789012345678901234567890123456789012345678901234567890
</td>
<td style="width:300px">
123456789012345678901234567890123456789012345678901234567890
</td>
</tr>
</table>

</body></html>
```

Mit `table-layout`: beeinflussen Sie die Tabellenanzeige bei Breitenangaben. Folgende Angaben sind erlaubt:

- `fixed` = Breitenangaben haben Vorrang vor dem Zelleninhalt
- `auto` = Zelleninhalt hat Vorrang vor Breitenangaben (Voreinstellung)

**Beachten Sie:** Netscape 6.x interpretiert diese Eigenschaft wohl, hatte aber Probleme bei der Darstellung. Eine `width`-Zuweisung an das `table`-Element brachte ihn dazu, die Anzeige zu verhauen, und das obige Beispiel mit der `width`-Zuweisung an einzelne Zellen, um damit die Breite der Tabelle festzulegen, ignorierte er.



### *border-collapse (Rahmenmodell)*

Mit dieser Eigenschaft können Sie festlegen, ob Einzelrahmen von Tabellenzellen zusammenfallen sollen oder nicht. Ein Beispiel:

```
<html><head><title>border-collapse</title>
</head><body bgcolor="FFFFFF" text="#000000">

<table border="0"><tr>
<th colspan="3">Normaleinstellung</th>
</tr><tr>
```

```

<td style="border:medium solid blue">Haus</td>
<td style="border:medium dashed blue">Auto</td>
<td style="border:medium dotted blue">Boot</td>
</tr></table>
<table border="0" style="border-collapse:collapse"><tr>
<th colspan="3">Spezialeinstellung (collapse)</th>
</tr><tr>
<td style="border:medium solid blue">Haus</td>
<td style="border:medium dashed blue">Auto</td>
<td style="border:medium dotted blue">Boot</td>
</tr></table>
</body></html>

```



16.34: Im unteren Beispiel sind die Trennlinien nebeneinander liegender Zellen zusammengefasst.

Mit `border-collapse`: beeinflussen Sie die Art, wie Einzelrahmen benachbarter Tabellenzellen reagieren. Folgende Angaben sind erlaubt:

- `separate` = Zellenrahmen fallen nicht zusammen
- `collapse` = Zellenrahmen fallen zusammen

Bei der Angabe `collapse` kommt es zu Konflikten zwischen Rahmenformatierungen einzelner Zellen. Rahmen, die dabei in jedem Fall Vorrang haben sollen und den gemeinsamen Grenzrahmen der Nachbarzelle überlagern sollen, müssen zu diesem Zweck die CSS-Angabe `border-style:hidden` erhalten. Rahmen, die in jedem Fall nachrangig sein sollen und von dem Grenzrahmen der Nachbarzelle überlagert werden sollen, müssen die CSS-Angabe `border-style:none` erhalten. Wenn keine der Zellen die Angabe `border-style:hidden` enthält, aber eine davon die Angabe `border-style:none`, entscheidet die Breitenangabe (`border-width`) zum Rahmen, welche Formatierung beim gemeinsamen Grenzrahmen Vorrang hat. Breitere Rahmen haben dann den Vorrang. Falls auch die Breitenangabe keine Konfliktlösung bringt, da die Breitenangaben gleich sind, dann entscheidet die folgende Reihenfolge von Rahmentypen (`border-style`): `double`, `solid`, `dashed`, `dotted`, `ridge`, `outset`, `groove`, `inset`.



|                     |                     |
|---------------------|---------------------|
| Zelle 1<br>separate | Zelle 2<br>separate |
| Zelle 1<br>collapse | Zelle 2<br>collapse |

16.35: Das Schaubild verdeutlicht das Zusammenlegen von angrenzenden Zellenrändern.

**Beachten Sie:** Netscape 6.(1) interpretiert diese Angabe noch nicht.

### **CSS 2.0** **N 6.0** *border-spacing (Rahmenabstand im Gitternetz)*

Mit `border-spacing`: können Sie für ein `table`-Element den Abstand der Zellenrahmen voneinander bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>border-spacing</title>
<style>
td { border:thin solid blue; }
</style>
</head><body bgcolor="FFFFFF" text="#000000">

<table style="border:thin solid red; border-spacing:10px">
<tr>
<td>Haus</td>
<td>Auto</td>
<td>Boot</td>
</tr></table>

</body></html>
```

**Beachten Sie:** Der Internet Explorer 5.x interpretiert diese Angabe noch nicht.

### **CSS 2.0** **N 6.0** *empty-cells (Anzeige/Nichtanzeige leerer Zellen)*

Mit dieser Eigenschaft können Sie festlegen, ob Tabellenzellen ohne Inhalt bei sichtbarem Gitternetz einen Rahmen erhalten sollen oder nicht. Ein Beispiel:

```
<html><head><title>empty-cells</title>
<style>
td { border:1px solid black; }
</style>
</head><body bgcolor="FFFFFF" text="#000000">

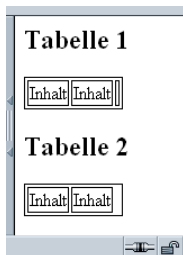
<h2>Tabelle 1</h2>
<table style="border:1px solid black; empty-cells:show"><tr>
<td>Inhalt</td><td>Inhalt</td><td></td>
</tr></table>
```

```

<h2>Tabelle 2</h2>
<table style="border:1px solid black; empty-cells:hide"><tr>
<td>Inhalt</td><td>Inhalt</td><td></td>
</tr></table>

</body></html>

```



16.36: In der oberen Tabelle ist die leere Zelle sichtbar, in der unteren nicht.

Mit `empty-cells`: bestimmen Sie, ob Rahmen leerer Tabellenzellen angezeigt werden oder nicht. Folgende Angaben sind erlaubt:

- `show` = Zellenrahmen leerer Tabellenzellen werden angezeigt
- `collapse` = Zellenrahmen leerer Tabellenzellen werden unterdrückt (Voreinstellung)

**Beachten Sie:** Der Internet Explorer 5.x interpretiert diese Angabe noch nicht.

### **CSS 2.0** *speak-header (Art der Sprachausgabeunterstützung)*

Wenn eine HTML-Tabelle nicht am Bildschirm angezeigt oder ausgedruckt, sondern von einem Sprecher oder Sprachausgabesystem vorgelesen wird (z.B. für Sehbehinderte), entsteht das Problem, dass ein zweidimensionaler Raum eindimensional in der Zeit gelesen werden soll. Mit der hier beschriebenen Angabe können Sie für diesen Fall entscheiden, ob der Inhalt von Kopfzellen bei jeder ihr untergeordneten Datenzeile erneut wiedergegeben werden soll oder nur einmal. Ein Beispiel:

```

<table border="1">
<tr>
<th style="speak-header-cell:always">Kopfzelle 1</th>
<th style="speak-header-cell:once">Kopfzelle 2</th>
</tr>
<tr>
<td>Datenzeile 1</td>
<td>Datenzeile 2</td>
</tr>
<tr>
<td>Datenzeile 3</td>
<td>Datenzeile 4</td>
</tr>
</table>

```

Mit `repeat-header-cell`: können Sie das Wiedergabeverhalten für Kopfzelleninhalte bestimmen. Das obige Beispiel bewirkt, dass die Kopfzeile 1 vor den Datenzellen 1 und 3 jeweils wiederholt wird, während die Kopfzeile 2 nur einmal ausgegeben wird. Folgende Angaben sind erlaubt:

- `always` = Kopfzelleninhalt bei jeder untergeordneten Datenzeile wiederholen
- `once` = Kopfzelleninhalt nur einmal wiedergeben (Normaleinstellung)

## 16.10 Pseudoformate

### Allgemeines zu Pseudoformaten

Was Pseudoformate sind, ist im Abschnitt 15.3.8 *Pseudoformate definieren* beschrieben.



**`:link`, `:visited`, `:hover`, `:active`, `:focus` (Pseudoformate für Verweise)**

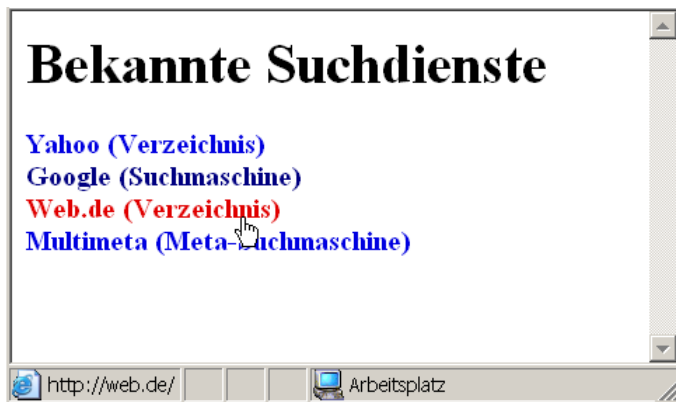
Sie können das Erscheinungsbild von Verweisen zu noch nicht besuchten Seiten, zu bereits besuchten Seiten und zu Verweisen, die gerade mit der Maus überfahren oder angeklickt werden, bestimmen. Ein Beispiel:

```
<html><head><title>:link, :visited, :hover, :active</title>
<style type="text/css">
a:link { font-weight:bold; color:#0000E0; text-decoration:none }
a:visited { font-weight:bold; color:#000080; text-decoration:none }
a:hover { font-weight:bold; color:#E00000; text-decoration:none }
a:active { font-weight:bold; color:#E00000; text-decoration:underline }
a:focus { font-weight:bold; color:#00E000; text-decoration:underline }
</style>
</head><body bgcolor="FFFFFF" text="#000000">

<h1>Bekannte Suchdienste</h1>
<p>
<a href="http://www.yahoo.de/">Yahoo (Verzeichnis)</a><br>
<a href="http://www.google.de/">Google (Suchmaschine)</a><br>
<a href="http://web.de/">Web.de (Verzeichnis)</a><br>
<a href="http://www.multimeta.de/">Multimeta (Meta-Suchmaschine)</a>
</p>

</body></html>
```

Pseudoformate werden zentral in einem `style`-Bereich notiert. Die hier beschriebenen Pseudoformate gelten für das `a`-Element in HTML, daher wird vor dem Doppelpunkt das `a` notiert. In den Formatdefinitionen für die einzelnen Pseudoformate können Sie beliebige, geeignete CSS-Eigenschaften zuweisen.



16.37: Beim Überfahren mit der Maus (:hover) erscheinen die Links rot.

Die Pseudoformate bedeuten:

- :link für Verweise zu noch nicht besuchten Seiten
- :visited für Verweise zu bereits besuchten Seiten
- :hover für Verweise, während der Anwender mit der Maus darüber fährt
- :active für gerade angeklickte Verweise
- :focus für Verweise, die den Fokus erhalten, z.B. durch »Durchsteppen« mit der Tabulatortaste

**Beachten Sie:** Um eine korrekte Darstellung dieser Pseudoformate zu erreichen, müssen Sie bei der Notierung die folgende Reihenfolge einhalten:

```
:link, :visited, :hover, :active
```

Netscape 4.x und der Internet Explorer 3.0 interpretieren noch nicht alle Angaben. Insbesondere reagiert Netscape 4.x noch nicht auf das Pseudoformat `a:hover`. Der Internet Explorer 5.x kennt die Angabe `a:focus` noch nicht.

`:focus` funktioniert bei anderen Elementen als Verweisen auch. Wenn Sie beispielsweise für `h1:focus` CSS-Eigenschaften definieren und dann mit der Maus auf eine `h1`-Überschrift klicken, nimmt diese, solange die Maus geklickt ist, die definierten Eigenschaften an.



***:first-line, :first-letter, :first-child (Pseudoformate für Absätze)***

Sie können für Absätze mit längerem Fließtext für die erste Zeile des Absatzes ein separates Erscheinungsbild erzwingen. Ferner können Sie für das erste Zeichen der ersten Zeile ein separates Erscheinungsbild erzwingen. Und schließlich können Sie sogar bestimmen, dass ein erstes Kindelement eines Elements ein besonderes Aussehen erhält. Dies sind typographische Effekte, die von Druckschriften her bekannt sind. Ein Beispiel:

```
<html><head><title>:first-line, :first-letter, :first-child</title>
<style type="text/css">
p:first-line { font-weight:bold }
```

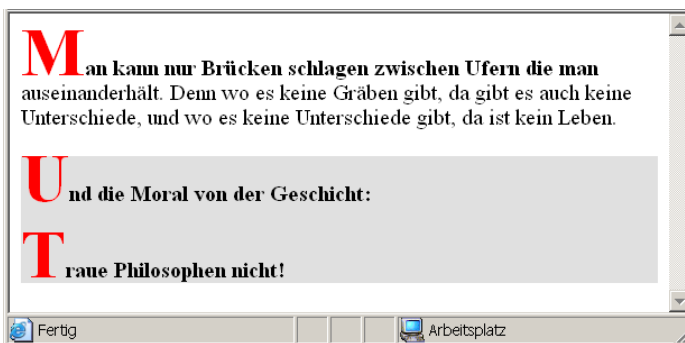
```

p:first-letter { font-size:300%; color:red }
div { background-color:#E0E0E0 }
div p:first-child { background-color:#C0C0C0; }

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

<p>Man kann nur Brücken schlagen zwischen Ufern die man auseinanderhält.
Denn wo es keine Gräben gibt, da gibt es auch keine Unterschiede, und wo es
keine Unterschiede gibt, da ist kein Leben.</p>
<div>
<p>Und die Moral von der Geschicht:</p>
<p>Traue Philosophen nicht!</p>
</div>
</body></html>

```



16.38: Pseudoformate für die erste Textzeile beziehungsweise den ersten Buchstaben.

Pseudoformate werden zentral in einem `style`-Bereich notiert. Die hier beschriebenen Pseudoformate sind für typische Absatz- oder Überschriftenelemente in HTML gedacht. In den Formatdefinitionen für die einzelnen Pseudoformate können Sie beliebige, geeignete CSS-Eigenschaften zuweisen. Die Pseudoformate bedeuten:

- `:first-line` = die erste Textzeile des Elements erhält die CSS-Eigenschaften.
- `:first-letter` = der erste Buchstabe des Textes erhält die CSS-Eigenschaften.
- `:first-child` = das erste Kindelement des Elements erhält die CSS-Eigenschaften.

**Beachten Sie:** Der Internet Explorer 5.x interpretiert `:first-child` noch nicht.



### *:before, :after (Pseudoformate für automatisch generierten Text)*

Sie können bestimmen, dass vor oder nach einem Element automatisch Inhalt eingefügt wird. Es kann sich um statischen Text handeln, aber auch um variable Inhalte. Ein Beispiel:

```

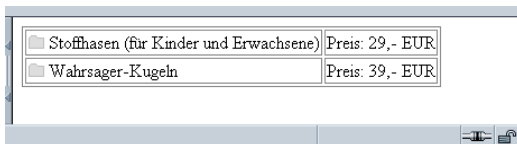
<html><head><title>:before, :after</title>
<style type="text/css">
td.Preis:before { content:"Preis: "; }
td.Preis:after { content:",- EUR"; }

```

```
td.Produkt:before { content:url(list_style_image.gif) " "; }
td.Produkt:after { content:" attr(title); }
</style>
</head><body bgcolor="FFFFFF" text="#000000">

<table border="1">
<tr>
<td class="Produkt" title="(f&uuml;r Kinder und Erwachsene)">Stoffhasen</td>
<td class="Preis">29</td>
</tr>
<tr>
<td class="Produkt">Wahrsager-Kugeln</td>
<td class="Preis">39</td>
</tr>
</table>

</body></html>
```



16.39: Die CSS-Angaben fügen ein Bildsymbol am Anfang bzw. die Währungskürzel am Ende ein.

Pseudoformate werden zentral in einem `style`-Bereich notiert. Die hier beschriebenen Pseudoformate bedeuten:

- `:before` = Inhalt, der vor dem notierten Inhalt des Elements eingefügt werden soll.
- `:after` = Inhalt, der nach dem notierten Inhalt des Elements eingefügt werden soll.

Innerhalb der geschweiften Klammern bestimmen Sie dann mit `content:`, was vor bzw. nach dem in HTML notierten Elementinhalt automatisch eingefügt werden soll.

Im obigen Beispiel wird für `td`-Elemente mit dem Klassennamen `Preis` bestimmt, dass vor dem Elementinhalt der statische Text `Preis:` ausgegeben werden soll und nach dem Elementinhalt der Text `EUR, -`. In HTML genügt es dann, nur die Zahlen zu notieren. Für Tabellenzellen mit dem Klassennamen `Produkt` wird bestimmt, dass vor dem Element eine Grafik eingefügt wird. Dazu können Sie die Syntax `url([URI])` zum Referenzieren der Grafik verwenden. Für die gleiche Tabellenzellenklasse wird festgelegt, dass wenn im einleitenden `<td>`-Tag ein Attribut `title=` notiert wird, dessen Wertzuweisung als Text hinter dem Elementinhalt eingefügt wird. Für diese Art von variablem Inhalt können Sie die Syntax `attr([Attributname])` verwenden, wie im Beispiel oben gezeigt.

Statischer Text hinter `content:` muss in Anführungszeichen stehen. Im obigen Beispiel können Sie auch sehen, dass es möglich ist, Angaben zu kombinieren. So bedeutet `content:url(list_style_image.gif) " ";`, dass hinter der Grafik noch ein statisches Leerzeichen eingefügt wird.

**Beachten Sie:** Diese Angaben werden vom Internet Explorer 5.x und 6.0 noch nicht interpretiert.

**CSS  
2.0****Automatische Nummerierung**

Eine erweiterte Syntax der Pseudoformate `:before` und `:after` erlaubt auch die automatische Nummerierung von Elementen, z.B. für Kapitelüberschriften. Ein Beispiel:

```
<html><head><title>:before, :after</title>
<style type="text/css">
h1:before { content:counter(Ebene01);
              counter-increment:Ebene01;
              counter-reset:Ebene02;
            }
h2:before { content:counter(Ebene01) ". " counter(Ebene02);
              counter-increment:Ebene02;
            }
</style>
</head><body bgcolor="FFFFFF" text="#000000">

<h1>&Uuml;berschrift erster Ordnung</h1>
<p>Text</p>
<h2>&Uuml;berschrift zweiter Ordnung</h2>
<p>Text</p>
<h2>&Uuml;berschrift zweiter Ordnung</h2>
<p>Text</p>
<h1>&Uuml;berschrift erster Ordnung</h1>
<p>Text</p>
<h2>&Uuml;berschrift zweiter Ordnung</h2>
<p>Text</p>
<h2>&Uuml;berschrift zweiter Ordnung</h2>
<p>Text</p>
<h2>&Uuml;berschrift zweiter Ordnung</h2>
<p>Text</p>
<h1>&Uuml;berschrift zweiter Ordnung</h1>
<p>Text</p>

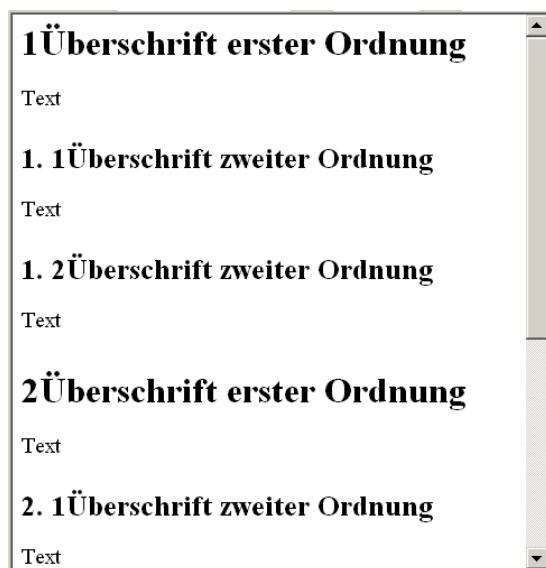
</body></html>
```

In der Regel werden Sie für die automatische Nummerierung ein Überschriftenelement wie `h1` oder `h2` angeben und die Nummerierung vor dessen Text setzen. Dann müssen Sie im `style`-Bereich Selektoren wie `h1:before` oder `h2:before` notieren. In den geschweiften Klammern folgt dann, wie bei `:before` und `:after` üblich, die Eigenschaft `content`. Dahinter können im Zusammenhang mit automatischer Nummerierung drei Funktionen folgen, die die Nummerierungszähler steuern. Trennen Sie mehrere Funktionsaufrufe durch Strichpunkte (;).

Mit `counter([Name])` fügen Sie den Wert des Zählers mit dem Namen `Name` an die aktuelle Stelle ein. Beim ersten Vorkommen hat der Zähler den Wert 1. In den runden Klammern notieren Sie Zählernamen. Im obigen Beispiel sind das die Namen `Ebene01` und `Ebene02`. Die Namen können Sie frei vergeben.

Mit `counter-increment:[Name]` zählen Sie den Zähler mit dem angegebenen Namen um 1 hoch. Normalerweise notieren Sie zuerst `counter([Name])` und dann `counter-increment:[Name]`, um zuerst den aktuellen Zählerstand einzufügen und dann den Zähler für das nächste Vorkommen des betreffenden Elementtyps hochzuzählen.

Mit `counter-reset:[Name]` setzen Sie einen Zähler wieder auf seinen Anfangswert (1) zurück. Dies ist vor allem wichtig, wenn Sie mit Kapitel-Unterkapitel-Nummerierung arbeiten, wie im obigen Beispiel. Dort wird beim Vorkommen eines neuen `h1`-Elements der Zähler für `h2`-Elemente wieder auf 1 gesetzt.



16.40: Die automatische Nummerierung in Opera 5.

Neben den Funktionen für die Zählerkontrolle kann auch statischer Text notiert werden. Im obigen Beispiel wird hinter dem Aufruf von `counter(Ebene01)` als statischer Text ein Punkt notiert. So entsteht im Beispiel durch die Verwaltung der beiden Zähler für Überschriften erster und zweiter Ordnung ein Nummerierungsschema wie 1, 1.1, 1.2, 2, 2.1, 2.2., 2.3, 3 usw.

**Beachten Sie:** Die automatische Nummerierung wird bislang weder von Netscape noch vom Internet Explorer unterstützt, wohl aber vom Opera-Browser.

## 16.11 Positionierung und Anzeige von Elementen

### *Allgemeines zur Positionierung und Anzeige von Elementen*

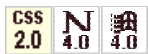
Ab der CSS-Version 2.0 gibt es verschiedene Eigenschaften, um Elemente einer WWW-Seite exakt im Anzeigefenster des WWW-Browsers zu positionieren und die Position von Elementen untereinander genau zu kontrollieren. Dazu gehören Angaben zum absoluten und relativen



Positionieren von Elementen, Angaben zur Ausdehnung von Elementen, Angaben zum Umfließen von Elementen und Angaben zum Überlappen und Anzeigen von Elementen.

Absolutes und relatives Positionieren von Elementen bietet Ihnen die Möglichkeit, das Erscheinungsbild von Web-Seiten stärker bildschirmorientiert zu gestalten. So können Sie für einzelne Bereiche festlegen, wo genau diese Bereiche beginnen sollen. Bereiche können sich überlappen usw. Im wesentlichen decken diese Stylesheet-Angaben die Möglichkeiten der alten Netscape-Layer ab (Kapitel 12). Im Gegensatz zu den Layern sind sie jedoch ein offizieller Vorschlag des W3-Konsortiums. Netscape und der Microsoft Internet Explorer interpretieren diese Angaben seit ihren 4er-Versionen – allerdings noch mit einigen Einschränkungen bzw. Besonderheiten.

Das Positionieren von Elementen ist eine wichtige Voraussetzung für viele Anwendungsfälle von Dynamischem HTML (Kapitel 21). Mit Hilfe entsprechender Script-Unterstützung können Sie positionierte Elemente bewegen, ein-/ausblenden, anders überlappen usw.



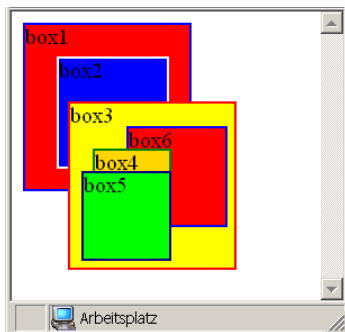
### **position (Positionsart)**

Sie können für einen Bereich bestimmen, wie er positioniert werden soll. Ein Beispiel:

```
<html><head><title>position</title>
<style type="text/css">
<!--
body { }
#box1 { position:absolute; top:10px; left:10px; width:150px; height:150px; z-
index:1; }
#box2 { position:absolute; top:40px; left:40px; width:100px; height:100px; z-
index:2; }
#box3 { position:absolute; top:80px; left:50px; width:150px; height:150px; z-
index:3; }
#box4 { position:absolute; top:40px; left:20px; width:70px; height:70px; z-index:1;
}
#box5 { position:absolute; top:60px; left:10px; width:80px; height:80px; z-index:2;
}
#box6 { position:absolute; top:20px; left:50px; width:90px; height:90px; z-index:-
1; }
-->
</style>
</head><body bgcolor="FFFFFF" text="#000000">

<div id="box1" style="background:red; border:solid 2px blue;">box1</div>
<div id="box2" style="background:blue; border:solid 2px white;">box2</div>
<div id="box3" style="background:yellow; border:solid 2px red;">box3
  <div id="box4" style="background:gold; border:solid 2px green;">box4</div>
  <div id="box5" style="background:lime; border:solid 2px navy;">box5</div>
  <div id="box6" style="background:red; border:solid 2px blue;">box6</div>
</div>

</body></html>
```



16.41: Das Beispiel erzeugt diese positionierten Bereiche.

Mit `position`: können Sie die Positionsart bestimmen. Folgende Angaben sind erlaubt:

|                       |                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------|
| <code>absolute</code> | Absolute Positionierung, gemessen am Rand des Elternelements. Scrollt mit.                        |
| <code>fixed</code>    | Absolute Positionierung, gemessen am Rand des Elternelements. Bleibt beim Scrollen stehen.        |
| <code>relative</code> | Relative Positionierung, gemessen an der Normalposition oder Anfangsposition des Elements selbst. |
| <code>static</code>   | Keine spezielle Positionierung, normaler Elementfluss (Normaleinstellung).                        |

Besonders das Verhalten der Angaben `absolute` und `relative` ist anfangs etwas verwirrend. Denn `absolute` verhält sich durchaus relativ, wie die inneren `div`-Elemente im obigen Beispiel zeigen: relativ nämlich zum Rand des Elternelements. Wenn sonst kein Elternelement existiert, ist das `body`-Element das Elternelement. Die Angabe `relative` bezieht sich dagegen auf die Normalposition des Elements selbst.

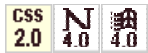
Die Angabe `position`: legt noch nicht fest, wo genau ein Element beginnen soll. Die Angabe macht nur Sinn, wenn Sie zugleich die gewünschte Startposition angeben. Dies können Sie beispielsweise mit Angaben zur `top` oder `left` tun (siehe die folgenden Abschnitte).

- Wenn Sie also beispielsweise `position:absolute; top:30px` angeben, dann legen Sie für das betreffende Element fest, dass es 30 Pixel unterhalb des Randes seines Elternelements beginnt.
- Wenn Sie `position:relative; top:5px` notieren, dann legen Sie für das Element fest, dass sein oberer Rand 5 Pixel tiefer liegt, als es normalerweise der Fall wäre.

**Beachten Sie:** Die Angabe `fixed` wird weder von Netscape noch vom MS Internet Explorer interpretiert. Weder Netscape noch der MS Internet Explorer interpretieren Angaben zur absoluten Positionierung in allen HTML-Elementen. Um eine maximale Abwärtskompatibilität zu erreichen, ist es am besten, Angaben zur Positionierung auf das `div`-Element zu beschränken. Schließen Sie andere Elemente, etwa Textabsätze, Grafiken, Tabellen, Formulare usw. einfach in entsprechende `div`-Elemente ein.

Netscape und der MS Internet Explorer reagieren bei absoluter Positionierung unterschiedlich, was die automatische Breite von Elementen betrifft. Dieser Fall tritt ein, wenn Sie eine Angabe für `left`: notieren und nicht angeben, wie breit der so definierte Bereich sein soll. Netscape

erstreckt das Element maximal bis zum rechten Fensterrand/Seitenrand, während der MS Internet Explorer es über den rechten Rand hinausdehnt (der Anwender muss quer scrollen). Bei relativer Positionierung reagieren beide Browser so wie der MS Internet Explorer bei der absoluten Positionierung. Um den Effekt zu vermeiden, können Sie z.B. Angaben zu `width` (Breite) des positionierten Bereichs machen.



### *top (Startposition von oben)*

Diese Angabe ist sinnvoll in Verbindung mit einer Angabe zu `position`. Sie können bestimmen, wo ein absolut oder relativ positioniertes Element von oben beginnt. Mit `top`: können Sie die Startposition von oben bestimmen. Erlaubt ist eine numerische Angabe oder der Wert `auto` für eine automatische Positionierung. Ein Beispiel:

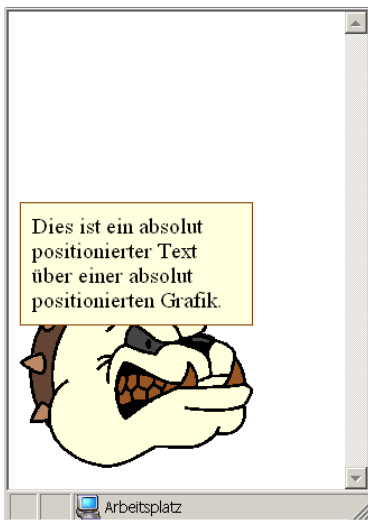
```
<html><head><title>top</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="position:absolute; top:4.8cm">

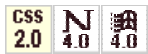
</div>

<div style="position:absolute; top:3.6cm; width:208px; background-color:#FFFFE0;
border:1px solid #804000; padding:10px">Dies ist ein absolut positionierter Text
&uuml;ber einer absolut positionierten Grafik.</div>

</body></html>
```



16.42: Bild und Bereich haben einen absoluten Abstand zum oberen Fensterrand.

**left (Startposition von links)**

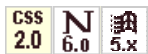
Diese Angabe ist sinnvoll in Verbindung mit einer Angabe zu `position`. Sie können bestimmen, wo ein absolut oder relativ positioniertes Element von links beginnt. Mit `left`: können Sie die Startposition von links bestimmen. Erlaubt ist eine numerische Angabe oder der Wert `auto` für eine automatische Positionierung. Ein Beispiel:

```
<html><head><title>left</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="position:absolute; top:100px; left:100px;">
</div>

<div style="position:absolute; top:140px;
  left:350px; width:200px;
  background-color:#FFFE0;
  border:1px solid #804000; padding:10px">
Dies ist ein absolut positionierter Text neben einer absolut positionierten
Grafik.</div>

</body></html>
```

**bottom (Startposition von unten)**

Diese Angabe ist sinnvoll in Verbindung mit einer Angabe zu `position`. Sie können bestimmen, wo ein absolut oder relativ positioniertes Element von unten beginnt. Mit `bottom`: können Sie die Startposition von unten bestimmen. Erlaubt ist eine numerische Angabe oder der Wert `auto` für eine automatische Positionierung. Ein Beispiel:

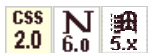
```
<html><head><title>bottom</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="position:absolute; bottom:0px; left:100px;">
Dies ist ein untenb&uuml;ndiger Hund:<br>

</div>

</body></html>
```

**Beachten Sie:** Die Browser der 4er-Generation interpretieren diese Eigenschaft noch nicht.

**right (Startposition von rechts)**

Diese Angabe ist sinnvoll in Verbindung mit einer Angabe zu `position`. Sie können bestimmen, wo ein absolut oder relativ positioniertes Element von rechts beginnt. Mit `right`: können Sie die Startposition von rechts bestimmen. Erlaubt ist eine numerische Angabe oder der Wert `auto` für eine automatische Positionierung.

Ein Beispiel:

```
<html><head><title>right</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="position:absolute; top:100px; right:0px;">
Rechtsbündiger Hund:<br>

</div>

</body></html>
```



16.43: Der Hund klebt am rechten Fensterrand.

**Beachten Sie:** Die Browser der 4er-Generation interpretieren diese Eigenschaft noch nicht.



### *width (Breite)*

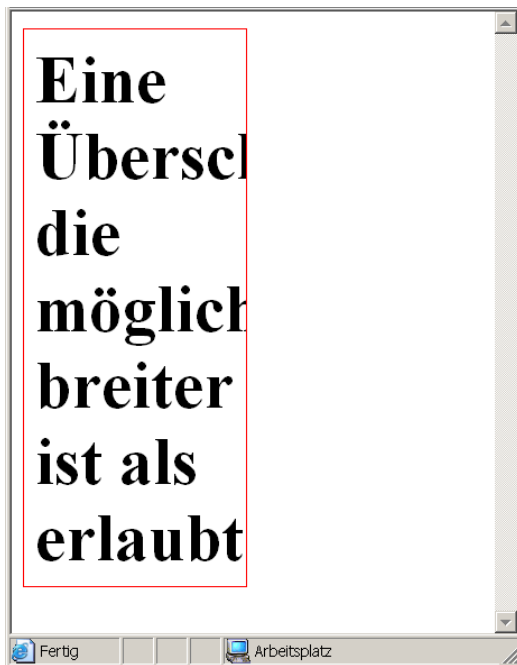
Sie können bestimmen, wie breit ein Element oder Bereich sein soll. Um zu bestimmen, was mit dem Inhalt passieren soll, wenn er mehr Breite einnimmt als die hier angegebene Breite, können Sie zusätzlich die Eigenschaft `overflow` verwenden. Mit `width:` können Sie die Breite bestimmen. Erlaubt ist eine numerische Angabe oder der Wert `auto` für automatische Breite. Ein Beispiel:

```
<html><head><title>width</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 style="width:200px; overflow:hidden; border:1px solid red; padding:10px; font-size:36pt">
Eine &Uuml;berschrift, die m&ouml;glicherweise breiter ist als erlaubt</h1>

</body></html>
```

**Beachten Sie:** Dass im obigen Beispiel wo nötig in der Breite Text abgeschnitten wird, liegt nicht an der Eigenschaft `width`, sondern an der Angabe `overflow:hidden`. Die Browser der 4er-Generation interpretieren diese Angabe jedoch noch nicht.



16.44: Die Breitenangabe für die Überschrift nimmt keine Rücksicht auf die Breite der Wörter.

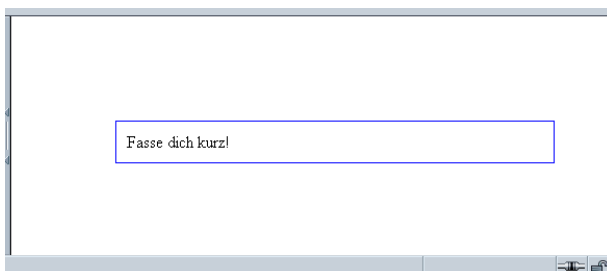
### **CSS 2.0** **N 6.0** *min-width (Mindestbreite)*

Sie können bestimmen, wie breit ein Element oder Bereich auf jeden Fall sein soll, auch wenn der Inhalt weniger Breite in Anspruch nimmt. Mit `min-width`: können Sie die Mindestbreite bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>min-width</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="position:absolute; top:100px; left:100px; min-width:400px; border:solid
1px blue; padding:10px">
Fasse dich kurz!</div>

</body></html>
```



16.45: Der Bereich wird so groß wie angegeben, auch wenn der Inhalt den Platz gar nicht benötigt.

**Beachten Sie:** Der Internet Explorer 5.x und 6.0 interpretiert diese Angabe noch nicht.

### **max-width (Maximalbreite)**

Sie können bestimmen, wie breit ein Element oder Bereich maximal sein soll. Um zu bestimmen, was mit dem Inhalt passieren soll, wenn er mehr vertikalen Raum einnimmt als die hier angegebene Höhe, können Sie zusätzlich die Eigenschaft `overflow` verwenden. Mit `max-width`: können Sie die Maximalbreite bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>max-width</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 style="max-width:250px; overflow:hidden; border:solid 1px green; font-
size:36pt; padding:10px">
Eine &Uuml;berschrift, bei das Wort 'm&ouml;l;glicherweise' m&ouml;l;glicherweise zu
breit f&uuml;r 250px ist</h1>

</body></html>
```

**Beachten Sie:** Der Internet Explorer 5.x und 6.0 interpretiert diese Angabe noch nicht.

### **height (Höhe)**

Sie können bestimmen, wie hoch ein Element oder Bereich sein soll. Um zu bestimmen, was mit dem Inhalt passieren soll, wenn er mehr vertikalen Raum einnimmt als die hier angegebene Höhe, können Sie zusätzlich die Eigenschaft `overflow` verwenden. Mit `height`: können Sie die Höhe bestimmen. Erlaubt ist eine numerische Angabe oder der Wert `auto` für automatische Höhe. Ein Beispiel:

```
<html><head><title>height</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 style="height:100px; width:300px; overflow:hidden; border:1px solid red;
padding:10px; font-size:36pt">
Eine &Uuml;berschrift, die m&ouml;l;glicherweise h&ouml;l;her ist als erlaubt</h1>

</body></html>
```

**Beachten Sie:** Dass im obigen Beispiel wo nötig in der Breite Text abgeschnitten wird, liegt nicht an der Eigenschaft `width`, sondern an der Angabe `overflow:hidden`. Die Browser der 4er-Generation interpretieren diese Angabe jedoch noch nicht.

### **min-height (Mindesthöhe)**

Sie können bestimmen, wie hoch ein Element oder Bereich auf jeden Fall sein soll. Mit `min-height`: können Sie die Mindesthöhe bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>min-height</title>
</head><body bgcolor="FFFFFF" text="#000000">
```

```
<div style="position:absolute; top:100px; left:100px; min-height:200px;
border:solid 1px blue; padding:10px">
Fasse dich kurz!</div>

</body></html>
```

**Beachten Sie:** Der Internet Explorer 5.x und 6.0 interpretiert diese Angabe noch nicht.

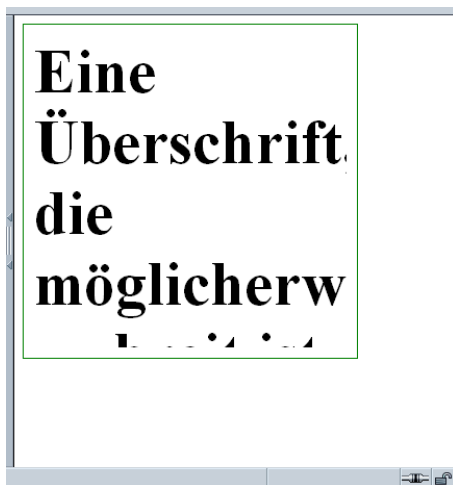
### **max-height (Maximalhöhe)**

Sie können bestimmen, wie hoch ein Element oder Bereich auf jeden Fall sein soll. Um zu bestimmen, was mit dem Inhalt passieren soll, wenn er mehr vertikalen Raum einnimmt als die hier angegebene Höhe, können Sie zusätzlich die Eigenschaft `overflow` verwenden. Mit `max-height`: können Sie die Maximalhöhe bestimmen. Erlaubt ist eine numerische Angabe. Ein Beispiel:

```
<html><head><title>max-height</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 style="max-width:300px; max-height:300px; overflow:hidden; border:solid 1px
green; font-size:36pt; padding:10px">
Eine &Uuml;berschrift, die m&ouml;glicherweise zu breit ist
f&uuml;r eine Box von 300 mal 300 Pixeln</h1>

</body></html>
```



16.46: Netscape 6.2 interpretiert die Maximalhöhe korrekt und schneidet den Text unten ab.

**Beachten Sie:** Der Internet Explorer 5.x und 6.0 interpretiert diese Angabe noch nicht.

### **overflow (Elementbereich mit übergroßem Inhalt)**

Diese Angabe kann von Bedeutung sein, wenn Sie für ein Element oder einen Bereich eine feste oder maximale gewünschte Breite oder Höhe definieren. Angenommen, Sie definieren einen



Bereich mit `<div>...</div>`, für den Sie mit Hilfe der Stylesheet-Angabe `width` eine Mindestbreite von 200 Pixeln erzwingen. Wenn Sie innerhalb dieses Bereichs eine Grafikreferenz (`<img>`) notieren, bei der das Bild eine Breite von mehr als 200 Pixeln hat, können Sie mit der hier beschriebenen Angabe regulieren, wie der Browser diesen Konflikt lösen soll. Ein Beispiel:

```
<html><head><title>overflow</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="position:absolute; top:100px;
  left:100px; width:200px;
  height:150px; overflow:hidden;
  border:thin solid #804000">
abgeschnittener Hund:<br>
</div>

</body></html>
```

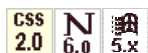


16.47: Mit `overflow` regeln Sie, was der Browser mit Inhalten macht, die zu groß für einen Bereich sind.

Mit `overflow`: können Sie bestimmen, wie übergroße innere Elemente behandelt werden. Folgende Angaben sind möglich:

| CSS-Angabe           | Wirkung                                                                                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>visible</code> | Element wird so weit ausgedehnt, dass sein Inhalt auf jeden Fall komplett sichtbar ist.                                                                                 |
| <code>hidden</code>  | Element wird abgeschnitten, wenn es die Grenzen überschreitet.                                                                                                          |
| <code>scroll</code>  | Element wird abgeschnitten, wenn es die Grenzen überschreitet. Der WWW-Browser sollte jedoch Scroll-Leisten anbieten, ähnlich wie in einem eingebetteten Frame-Fenster. |
| <code>auto</code>    | Der Web-Browser soll entscheiden, wie das Element im Konfliktfall angezeigt wird. Auch das Anbieten von Scroll-Leisten soll dabei erlaubt sein.                         |

**Beachten Sie:** Netscape 4.x kennt diese Eigenschaft nicht. Der Internet Explorer unterstützt die Angabe `overflow:visible` nicht.



### **direction (Richtung)**

Mit dieser Angabe können Sie die Schreibrichtung bei Elementen erzwingen und abhängig davon auch, in welche Richtung beispielsweise Breitenangaben berechnet werden oder an welcher Seite überbreite Inhalte mit `overflow` abgeschnitten werden. Die Schreibrichtung ist vor allem für die Darstellung von Schriftkulturen wie der arabischen gedacht, wo von rechts nach links geschrieben wird. Ein Beispiel:

```
<html><head><title>direction</title>
</head><body bgcolor="FFFFFF" text="#000000" style="direction:rtl">

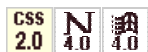
<h1>Hi Junkie!</h1>
<p>Hier kommt nnteglicherweise alles anders aussehen als normal ...</p>

</body></html>
```

Mit `direction`: können Sie die Schreibrichtung bestimmen. Folgende Angaben sind möglich:

- `ltr` = von links nach rechts
- `rtl` = von rechts nach links

**Beachten Sie:** Die Browser der 4er-Generation interpretieren diese Angabe noch nicht.



### **float (Textumfluss)**

Sie können für einen Bereich oder ein Element bestimmen, dass nachfolgende Elemente diesen Bereich bzw. dieses Element umfließen. Ein Beispiel:

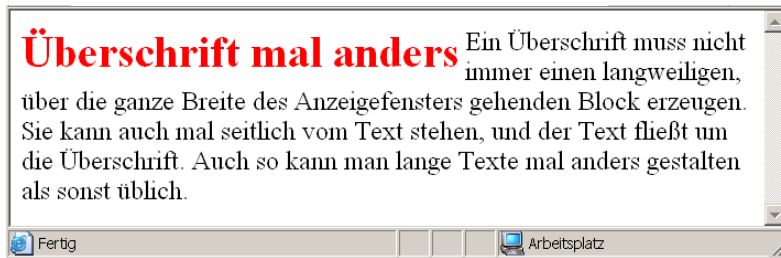
```
<html><head><title>float</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 style="width:400px; float:left; color:red">Umläberschrift mal anders</h1>
<p style="font-size:120%">Ein Umläberschrift muß nicht immer einen
langweiligen, über die ganze Breite des Anzeigefensters gehenden Block
erzeugen. Sie kann auch mal seitlich vom Text stehen, und der Text fließt um
die Umläberschrift. Auch so kann man lange Texte mal anders gestalten als sonst
möglich.</p>

</body></html>
```

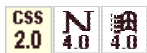
Mit `float`: können Sie bestimmen, dass nachfolgende Elemente das aktuelle Element bzw. den aktuellen Bereich umfließen. Folgende Angaben sind möglich:

- `left` = Element steht links und wird rechts davon von nachfolgenden Elementen umflossen.
- `right` = Element steht rechts und wird links davon von nachfolgenden Elementen umflossen.
- `none` = Kein Umfluss (Normaleinstellung).



16.48: Der Text fließt rechts um die Überschrift.

**Beachten Sie:** Wenn Sie für ein Element eine Angabe zu `float` notieren, müssen Sie auch eine Angabe zu `width` definieren.



*clear (Fortsetzung bei Textumfluss)*

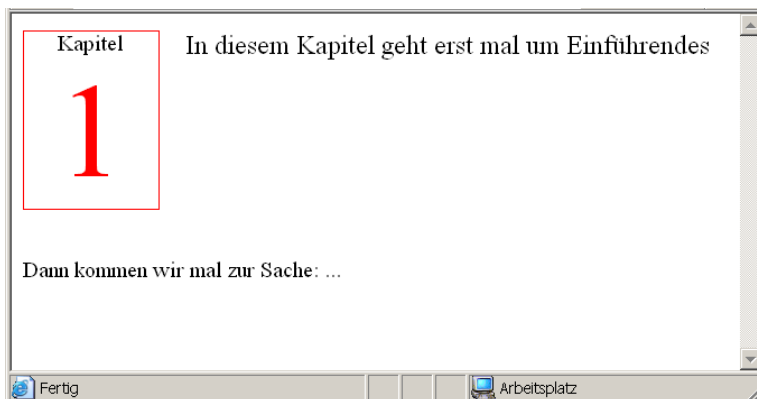
Wenn Sie mit `float` einen Textumfluss definieren, möchten Sie vielleicht nur zwei oder drei Zeilen kommentierenden Text neben einem umflossenen Element haben und den folgenden Text dann unterhalb des umflossenen Elements fortsetzen. Dazu dient die hier beschriebene Angabe. Ein Beispiel:

```
<html><head><title>clear</title>
</head><body bgcolor="FFFFFF" text="#000000">

<div style="float:left; margin-right:20px; margin-bottom:20px;
border:solid 1px red; text-align:center">
Kapitel<br>
<span style="font-family:Algerian,serif; font-size:72pt; color:red">
&nbsp;   &nbsp;  &nbsp;  </span></div>

<p style="font-size:120%">In diesem Kapitel geht erst mal um Einf&uuml;hrendes</p>
<p style="clear:left">Dann kommen wir mal zur Sache: ...</p>

</body></html>
```

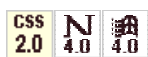


**16.49:** Mit `clear`: setzen Sie den Textfluss unterhalb eines Elements fort.

Mit `clear` können Sie einen Umfluss abbrechen und die Fortsetzung unterhalb des umflossenen Elements oder Bereichs erzwingen. Folgende Angaben sind möglich:

| CSS-Angabe         | Wirkung                                                          |
|--------------------|------------------------------------------------------------------|
| <code>left</code>  | Erzwingt bei <code>float:left</code> die Fortsetzung unterhalb.  |
| <code>right</code> | Erzwingt bei <code>float:right</code> die Fortsetzung unterhalb. |
| <code>both</code>  | Erzwingt in jedem Fall die Fortsetzung unterhalb.                |
| <code>none</code>  | Erzwingt keine Fortsetzung unterhalb (Normaleinstellung).        |

Im obigen Beispiel werden die Eigenschaften `float` und `clear` dazu genutzt, eine seitliche, große Kapitelnummer zu notieren, neben der Kapitelnummer einen kleinen Text, worum es in dem Kapitel geht, und unterhalb dieser beiden Elemente dann den normalen Text des Kapitels.



### **z-index (Schichtposition bei Überlappung)**

Diese Angabe ist sinnvoll in Verbindung mit mehreren Angaben zu `position`. Wenn Sie mehrere Elemente absolut positionieren, deren Anzeigebereiche sich überlappen, werden die Elemente normalerweise in der Reihenfolge übereinander angezeigt, in der sie definiert werden. Sie können die Reihenfolge ändern, indem Sie für die einzelnen Elemente Nummern vergeben. Elemente mit höherer Nummer überdecken Elemente mit niedrigerer Nummer. Ein Beispiel:

```
<html><head><title>z-index</title>
</head><body bgcolor="FFFFFF" text="#000000">

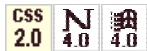
<div style="position:absolute; top:100px; left:100px; z-index:1; border:thin solid
#804000;">
<b>1.</b></div>
<div style="position:absolute; top:130px; left:150px; z-index:4; border:thin solid
#804000;">
<b>2.</b></div>
<div style="position:absolute; top:160px; left:200px; z-index:2; border:thin solid
#804000;">
<b>3.</b></div>
<div style="position:absolute; top:190px; left:250px; z-index:3; border:thin solid
#804000;">
<b>4.</b></div>

</body></html>
```

Mit `z-index` können Sie die Reihenfolge von überlappenden Elementen bestimmen. Notieren Sie für jedes Element, für das Sie eine eindeutige Schichtposition festlegen wollen, eine Zahl. Je höher die Zahl, desto weiter vorne das Element, je niedriger, desto weiter hinten das Element.

Im obigen Beispiel wird vier Mal der gleiche Bereich erzeugt – mit einem transparenten GIF-Bild als Inhalt. Die Bereiche überlappen sich aufgrund ihrer Angaben zur Positionierung und den Ausmaßen des Bildes. Normalerweise überlagert der letzte notierte Bereich alle anderen, da er als

letzter notiert ist. Im Beispiel überlagert jedoch der zweite Bereich alle anderen, weil er die höchste `z-index`-Nummer erhält.



### ***display (Anzeigeart bzw. Nichtanzeige ohne Platzhalter)***

Sie können bestimmen, ob ein Element zunächst angezeigt werden soll oder nicht. »Ohne Platzhalter« bedeutet, dass bei Nichtanzeige des Elements auch kein Raum für das Element reserviert wird. In diesem Fall deutet bei der Anzeige also zunächst nichts darauf hin, dass das Element überhaupt existiert. Mit Hilfe von Dynamischem HTML (Kapitel 21) können Sie so ein Element nachträglich anzeigen oder wieder verstecken.

Ferner können Sie mit der hier beschriebenen Angabe bestimmen, wie ein Element angezeigt werden soll. Sie können bestimmen, dass es einen eigenen (absatzerzeugenden) Block bilden oder dass es mitten im Text (inline) angezeigt werden soll. So können Sie beispielsweise absatzerzeugende Elemente wie Überschriften in Elemente umdefinieren, die keinen eigenen Absatz erzeugen. Ein Beispiel:

```
<html><head><title>display</title>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 style="display:inline">Ich bin ein inline-Element</h1>
&nbsp;<p style="display:inline">Dieser Text beweist es!</p>
<h1 id="versteckt" style="display:none">Ich bin ein Blockelement, werde aber nicht
angezeigt</h1>
<p>Und dieser Text tut so als ob sonst nichts w&auml;re!</p>

</body></html>
```

Mit `display`: können Sie die Anzeige von Elementen unterdrücken oder die Art der Anzeige festlegen. Folgende Angaben sind möglich:

| CSS-Angabe                                   | Wirkung                                                                                                |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>block</code>                           | Erzwingt einen Block – das Element erzeugt eine neue Zeile.                                            |
| <code>inline</code>                          | Erzwingt die Anzeige im Text – das Element wird im laufenden Textfluss angezeigt.                      |
| <code>list-item</code>                       | Wie <code>block</code> , jedoch mit einem Aufzählungszeichen (Bullet) davor.                           |
| <code>marker</code>                          | Deklariert automatisch generierten Text für das Element.                                               |
| <code>run-in</code> und <code>compact</code> | Bewirken, dass das Element kontext-abhängig als Blockelement oder als Inline-Element dargestellt wird. |
| <code>none</code>                            | Element wird nicht angezeigt und es wird auch kein Platzhalter freigelassen.                           |

Speziell für die Erzeugung von Tabellen mit Nicht-Tabellen-Elementen stehen weitere Angaben zu `display` zur Verfügung. Die entsprechenden Angaben lassen sich allerdings auch auf entsprechende HTML-Tabellenelemente anwenden. Gedacht sind sie aber eher für die tabellarische Darstellung von XML-Daten. Denn XML kennt kein Handling, um Daten als tabellarisch auszuzeichnen. Das muss die Style-Sprache übernehmen, mit deren Hilfe die XML-Daten angezeigt werden. In CSS hat man die `display`-Eigenschaft zu diesem Zweck auserkoren. Folgende Angaben sind in diesem Zusammenhang möglich:

| CSS-Angabe                      | Wirkung                                                                                                                                                                                                                      |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>table</code>              | Das Element enthält tabellarisch angeordnete Kindelemente und erzeugt eine neue Zeile. Wirkt wie das <code>table</code> -Element in HTML.                                                                                    |
| <code>inline-table</code>       | Das Element enthält tabellarisch angeordnete Kindelemente und wird im laufenden Textfluss angezeigt. Wirkt wie das <code>table</code> -Element in HTML, aber inline.                                                         |
| <code>table-row</code>          | Das Element enthält nebeneinander angeordnete Kindelemente. Wirkt wie das <code>tr</code> -Element in HTML.                                                                                                                  |
| <code>table-row-group</code>    | Das Element enthält eine Gruppe von Elementen mit nebeneinander angeordneten Kindelementen. Wirkt wie das <code>tbody</code> -Element in HTML.                                                                               |
| <code>table-header-group</code> | Das Element enthält eine Gruppe von Elementen mit nebeneinander angeordneten Kindelementen. Wirkt wie das <code>thead</code> -Element in HTML.                                                                               |
| <code>table-footer-group</code> | Das Element enthält eine Gruppe von Elementen mit nebeneinander angeordneten Kindelementen. Wirkt wie das <code>tfoot</code> -Element in HTML.                                                                               |
| <code>table-column</code>       | Das Element steht für eine Gruppe von Elementen, die eine Spalte der Tabelle bilden. Wirkt wie das <code>col</code> -Element in HTML.                                                                                        |
| <code>table-column-group</code> | Das Element steht für eine Gruppe von Elementen mit der Eigenschaft <code>table-cell</code> . Das Element steht für eine Tabellenzelle. Wirkt wie das <code>td</code> -Element in HTML (auch für <code>th</code> -Elemente). |
| <code>table-caption</code>      | Das Element steht für eine Tabellenüberschrift. Wirkt wie das <code>caption</code> -Element in HTML (auch für <code>th</code> -Elemente).                                                                                    |

#### Ein Beispiel:

```
<html><head><title>display</title>
<style type="text/css">
div.table { display:table; border-collapse:collapse; }
div.tr { display:table-row; }
div.td { display:table-cell; border:thin solid red; padding:5px; }
</style>
</head><body bgcolor="FFFFFF" text="#000000">

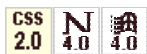
<div class="table">
  <div class="tr">
    <div class="td">ich</div>
    <div class="td">bin</div>
    <div class="td">eine</div>
    <div class="td">Tabelle</div>
  </div>
</div>

</body></html>
```

Das Beispiel definiert Klassen für `div`-Elemente. Die Klassennamen heißen genauso wie bekannte HTML-Tabellenelemente. Das muss nicht so sein und ist hier nur zu Demonstrationszwecken so. Die jeweiligen Klassen erhalten mit Hilfe von `display` Eigenschaften, durch die sich die Elemente, denen sie zugewiesen werden, wie die entsprechenden HTML-Elemente für

Tabellen verhalten. Innerhalb des Dateikörpers wird dann im Beispiel eine waschechte Tabelle aus `div`-Elementen mit `class`-Attributen zusammengebaut.

**Beachten Sie:** Netscape 4.0 und der MS Internet Explorer 4.0 interpretieren die Angabe `display:none` korrekt. Die übrigen Angaben werden von den 4er-Versionen der beiden Browser jedoch nur unvollständig oder noch gar nicht interpretiert. Die Angaben zu den Tabelleneigenschaften werden von Netscape 6.0 interpretiert, vom Internet Explorer 5.x und 6.0 noch nicht.



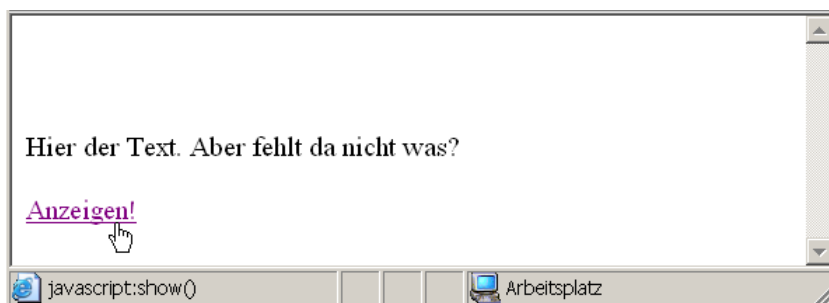
### visibility (Anzeige bzw. Nichtanzeige mit Platzhalter)

Sie können bestimmen, ob ein Element zunächst angezeigt werden soll oder nicht. »Mit Platzhalter« bedeutet, dass bei Nichtanzeige des Elements trotzdem Raum für das Element reserviert wird. Mit Hilfe von Dynamischem HTML (Kapitel 21) können Sie so ein Element nachträglich anzeigen oder wieder verstecken. Ein Beispiel:

```
<html><head><title>visibility</title>
<script language="JavaScript" type="text/javascript">
<!--
function show() {
  if(document.getElementById)
    document.getElementById("Ueberschrift").style.visibility = "visible";
}
//-->
</script>
</head><body bgcolor="FFFFFF" text="#000000">

<h1 id="Ueberschrift" style="visibility:hidden">Die &Uuml;berschrift zum Text</h1>
<p>Hier der Text. Aber fehlt da nicht was?</p>
<p><a href="javascript:show()">Anzeigen!</a></p>

</body></html>
```

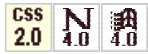


16.50: Erst ein Mausklick startet das Script, das die Überschrift sichtbar macht.

Mit `visibility`: können Sie bestimmen, ob ein Element anfangs angezeigt wird oder nicht. Folgende Angaben sind möglich:

- `hidden` = Der Inhalt des Element wird zunächst versteckt (nicht angezeigt).
- `visible` = Der Inhalt des Element wird zunächst angezeigt (Normaleinstellung).

Im obigen Beispiel wird die Überschrift zunächst nicht angezeigt. Beim Anklicken des Verweises wird dies jedoch dynamisch nachgeholt. Auf die dazu erforderlichen JavaScript-Befehle wird an dieser Stelle nicht näher eingegangen.



### *clip (Anzeigebereich eingrenzen)*

Sie können bestimmen, dass nur ein Ausschnitt eines Elements sichtbar angezeigt wird, unabhängig davon, welche Angaben sonst zur Größe des Elements gemacht wurden und unabhängig von der automatischen Größe des Elements. Wenn das Element größer ist als der so definierte Anzeigebereich, wird es an den Seiten entsprechend abgeschnitten. Derzeit können Sie nur ein Rechteck für den Ausschnitt definieren. In Zukunft sollen jedoch auch Polygone, Kreise bzw. Ellipsen möglich sein. Ein Beispiel:

```
<html><head><title>clip</title>
</head><body>

<div style="position:absolute; top:100px; left:100px; clip:rect(0px 130px 130px
0px);">
Etwas Hund:<br>
</div>

</body></html>
```

Mit `clip`: können Sie einen Ausschnitt für die sichtbare Anzeige definieren. Dahinter folgt der Bezeichner `rect` (für `rectangle` = Rechteck) und dahinter, in runde Klammern eingeschlossen, vier numerische Werte oder das Schlüsselwort `auto` zur Bestimmung des Ausschnitts.

- Der erste der vier Werte bezeichnet den Wert für »oben«, gemessen an der oberen Elementgrenze.
- Der zweite der vier Werte bezeichnet den Wert für »rechts«, gemessen an der linken Elementgrenze.
- Der dritte der vier Werte bezeichnet den Wert für »unten«, gemessen an der oberen Elementgrenze.
- Der vierte der vier Werte bezeichnet den Wert für »links«, gemessen an der linken Elementgrenze.

Damit überhaupt ein Ausschnitt sichtbar ist, muss der Wert für »unten« größer sein als der für »oben« und der Wert für »rechts« größer als der für »links«. Das Schlüsselwort `auto` bedeutet: keine Angabe zu dem betreffenden Wert, das Element soll an der entsprechenden Seite bis zu seiner Elementgrenze angezeigt werden.

**Beachten Sie:** Sowohl bei Netscape als auch beim Internet Explorer war die `clip`-Eigenschaft nur im Zusammenhang mit absolut positionierten Elementen nachvollziehbar.



## 16.12 Layouts für Printmedien

### Allgemeines zu Seitenlayout und Seitenumbruch

Die hier beschriebenen Stylesheet-Angaben eröffnen HTML und seiner Ergänzungssprache CSS völlig neue Räume, denn mit diesen Angaben wird es möglich, Drucklayouts zu definieren. Die Möglichkeit des zweigleisigen Editierens wird auch durch die Möglichkeit unterstützt, verschiedene Stylesheets für unterschiedliche Ausgabemedien zu definieren, wie im Abschnitt 15.2 *CSS Stylesheets in HTML einbinden* beschrieben.

Die Stylesheet-Angaben zu Seitenlayout und Seitenumbruch sind jedoch nicht nur für Druckerzeugnisse interessant. Ein entsprechendes Präsentationswerkzeug bzw. ein entsprechend ausgerichteter Web-Browser könnte diese Angaben auch zur bildschirmseitenorientierten Präsentation benutzen.

Zur Zeit ist das alles jedoch noch Zukunftsmusik. Die Stylesheet-Befehle liegen vor, doch die Browser unterstützen sie noch nicht. Von den Befehlen interpretiert lediglich der MS Internet Explorer seit Version 4.x diejenigen für den Seitenumbruch.

#### **CSS 2.0** @page (Seitenlayout definieren)

Ein Seitenlayout wird in einem zentralen `style`-Bereich definiert. Ein Beispiel:

```
<style type="text/css">
  @page { /* Angaben zum Seitenlayout */ }
</style>
```

Mit `@page` können Sie ein Seitenlayout erstellen. Die Angabe ist ein Behälter für die einzelnen Angaben zum Aussehen des Layouts. Dazu gehören Eigenschaften wie `size` (Seitengröße), `margin` (Seitenränder) und bei Bedarf auch Angaben zu `marks` (Registermarken). Diese Angaben stehen in den geschweiften Klammern im Anschluss an `@page`.

**Beachten Sie:** `@page` stellt sicher, dass sich die Definitionen zum Seitenlayout in den geschweiften Klammern nur auf »seitenorientierte« Medien beziehen und vom WWW-Browser nicht irrtümlich als Definitionen für das normale Erscheinungsbild der HTML-Datei am Bildschirm interpretiert werden. So können Sie mit dieser Angabe z.B. völlig unabhängig von sonstigen Stylesheet-Angaben im gleichen Bereich ein Seitenlayout für den Ausdruck Ihrer Web-Seiten erstellen.

#### **CSS 2.0** size (Seitengröße)

Sie können bestimmen, welche Breite und Höhe Ihr Seitenlayout haben soll. Für Druckdokumente definieren Sie auf diese Weise das gewünschte Papierformat. Für bildschirmseitenorientierte Präsentationen können Sie die Größe aller Bildschirmseiten festlegen. Ein Beispiel:

```
<html><head><title>size</title>
<style type="text/css">
  @page { size:landscape; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">
```

```
<p>Diese Seite erscheint an Ihrem Bildschirm genau so wie sonst. Wenn Ihr Browser
jedoch die Stylesheet-Angaben zum Seitenlayout kennt, versuchen Sie mal, diese
Datei &uuml;ber den normalen Druckbefehl Ihres Browsers auszudrucken.</p>

</body></html>
```

Mit `size`: können Sie im Anschluss an `@page` die Seitengröße definieren. Erlaubt sind zwei numerische Angaben, durch ein Leerzeichen voneinander getrennt. Dabei bedeutet die erste Angabe die Breite der Seite (z.B. 21.0cm) und die zweite Angabe die Höhe der Seite (z.B. 29.7cm). Ferner sind folgende Angaben erlaubt:

- `auto` = Das normale Format des Ausgabemediums (Normaleinstellung).
- `landscape` = Das normale Format des Ausgabemediums, bei rechteckigem Format jedoch querkant.
- `portrait` = Das normale Format des Ausgabemediums, bei rechteckigem Format jedoch hochkant.

### **CSS 2.0** *margin (Seitenränder)*

Sie können die Abstände zwischen Seitenrand und Seiteninhalt festlegen. Ein Beispiel:

```
<html><head><title>margin (page)</title>
<style type="text/css">
@page { size:21.0cm 14.85cm;
        margin-top:1.7cm;
        margin-bottom:1.4cm;
        margin-left:2cm;
        margin-right:2cm
    }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Diese Seite erscheint an Ihrem Bildschirm genau so wie sonst. F&uuml;r den
Ausdruck wurden jedoch Abst&auml;nde zwischen Seiteninhalt und Seitenrand
festgelegt.</p>

</body></html>
```

Um die Seitenränder einzeln zu definieren, stehen folgende Angaben zur Verfügung:

| CSS-Angabe                 | Wirkung            |
|----------------------------|--------------------|
| <code>margin-top</code>    | oberer Seitenrand  |
| <code>margin-bottom</code> | unterer Seitenrand |
| <code>margin-left</code>   | linker Seitenrand  |
| <code>margin-right</code>  | rechter Seitenrand |

Wenn Sie für alle vier Seitenränder einen einheitlichen Wert definieren wollen, können Sie anstelle dieser Angaben auch die Angabe `margin` verwenden. Für alle Seitenrandangaben ist eine numerische Angabe erlaubt.

**Beachten Sie:** Diese Angaben haben keinen Einfluss auf Seitenränder bei der normalen Browser-Präsentation am Bildschirm, da sie hinter der Seitenlayout-Angabe `@page` stehen. Für die normale HTML-Präsentation im WWW-Browser gibt es entsprechende Stylesheet-Angaben zu Abstand/Rand.

### **CSS 2.0** *marks (Schnitt- und Passermarken)*

Schnittmarken (engl. *crop marks*) sind Hilfslinien, die bei der Weiterverarbeitung in Belichtung und Druck zum Zuschneiden von Seiten benutzt werden. Passermarken, auch Registermarken (engl. *cross marks* oder *register marks*), sind Markierungen zum exakten Ausrichten des Satzspiegels bei Belichtung und Druck.

Wenn Sie also beispielsweise ein Seitenformat von 10.8 cm mal 18.2 cm definieren und dieses auf DIN-A-4-Papier ausdrucken, können Sie solche Marken setzen, die sichtbar im Ausdruck erscheinen und für die Satzspiegelvorbereitung in der Belichtung oder in der Druckerei das genaue Seitenformat anzeigen. Ein Beispiel:

```
<html><head><title>marks</title>
<style type="text/css">
@page { size:11.3cm 19.85cm; margin:1.7cm; marks:cross; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Diese Seite erscheint an Ihrem Bildschirm genau so wie sonst. F&uuml;r
Printmedienverarbeitung wurden jedoch Passermarken gesetzt.</p>

</body></html>
```

Mit `marks`: können Sie im Anschluss an `@page` Schnitt- und Passermarken definieren. Folgende Angaben sind erlaubt:

- `none` = keine Marken (Normaleinstellung)
- `crop` = Schnittmarken setzen
- `cross` = Passermarken setzen

Die Angaben `crop` und `cross` können Sie beide angeben, durch Leerzeichen getrennt. Dann werden beide Markentypen berücksichtigt.

### **CSS 2.0** *Seiten rechts/links und erste Seite*

Sie können für linke und rechte Seiten sowie für die erste Dokumentseite unterschiedliche Angaben notieren. Dadurch können Sie beispielsweise Heft- oder Binderänder ausgleichen.

Ein Beispiel:

```
<html><head><title>Seiten rechts/links</title>
<style type="text/css">
  @page { size:21.0cm 14.85cm; margin-top:1.5cm; margin-bottom:2cm }
  @page :left { margin-left:1.5cm; margin-right:2cm }
  @page :right { margin-left:2cm; margin-right:1.5cm }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Das hier ist die eine Seite.</p>
<p style="page-break-before:always">Das hier die andere (wegen erzwungenem
Seitenumbruch)</p>
<p>Die beiden Seiten haben unterschiedliche Seitenr&auml;nder.</p>

</body></html>
```

Unterschiedliche Seiten für rechts und links erzeugen Sie, indem Sie hinter @page, getrennt durch ein Leerzeichen, einen Doppelpunkt notieren und sofort dahinter eines der Schlüsselwörter *left* (zur Bezeichnung linker Seiten), *right* (zur Bezeichnung rechter Seiten) oder *first* (zur Bezeichnung der ersten Seite) angeben. Dahinter können Sie in geschweiften Klammern Angaben zu Größe, Seitenrändern und Schnitt-/Passermarken für die beiden Seitentypen notieren. In der Praxis ist die Unterscheidung vor allem nützlich, um (wie im obigen Beispiel) spiegelverkehrte Werte zum rechten und linken Seitenrand anzugeben. So lassen sich Heft-/Binderänder ausgleichen.



### ***page-break-before (Seitenumbruch vor einem Element)***

Sie können einen Seitenumbruch **vor** einem HTML-Element erzwingen oder verhindern.

Ein Beispiel:

```
<html><head><title>page-break-before</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<h1>Die erste Seite</h1>
<p>und ihr Text</p>
<h1 style="page-break-before:always">Und die zweite Seite</h2>
<p>und ihr Text</p>

</body></html>
```

Mit *page-break-before*: können Sie einen Seitenumbruch vor dem aktuellen Element erzwingen oder auch verhindern. Folgende Angaben sind erlaubt:

| CSS-Angabe | Wirkung                                                                                       |
|------------|-----------------------------------------------------------------------------------------------|
| always     | Immer einen Seitenumbruch vor dem aktuellen Element einfügen.                                 |
| avoid      | Nie einen Seitenumbruch vor dem aktuellen Element einfügen.                                   |
| left       | Bei linken/rechten Seiten Seitenumbruch einfügen, aktuelles Element auf nächste linke Seite.  |
| right      | Bei linken/rechten Seiten Seitenumbruch einfügen, aktuelles Element auf nächste rechte Seite. |
| inherit    | Angabe zum Seitenumbruch vom Elternelement übernehmen (erben).                                |
| auto       | Keine Angabe zum Seitenumbruch (Normaleinstellung).                                           |



### **page-break-after (Seitenumbruch nach einem Element)**

Sie können einen Seitenumbruch **nach** einem HTML-Element erzwingen oder verhindern. Ein Beispiel:

```
<html><head><title>page-break-after</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<h1>Die erste Seite</h1>
<p style="page-break-after:avoid; height:22cm; border:thin solid black;
padding:0.5cm">Leerraum</p>
<p>Kommentar dazu</p>
<h1>Neue &Uuml;berschrift</h1>
<p>und ihr Text</p>

</body></html>
```

Mit `page-break-after`: können Sie einen Seitenumbruch vor dem aktuellen Element erzwingen oder auch verhindern. Es sind dieselben Angaben wie bei `page-break-before`: erlaubt, nur beziehen sie sich auf das nachfolgende Element.



### **orphans (allein stehende Zeilen am Seitenende)**

Sie können verhindern, dass Seitenumbrüche innerhalb von Fließtext den Text so unglücklich trennen, dass eine einzelne Zeile auf der einen Seite steht und alle anderen Absatzzeilen auf die nächsten Seite kommen (DTP-Fachjargon hierzulande: *Schusterjungen*). Mit `orphans`: bestimmen Sie, wie viele Zeilen eines längeren Fließtextabsatzes auf der Seite vor dem Umbruch mindestens stehen sollen. Erlaubt ist eine Zahl für die Anzahl Zeilen. Voreinstellung ist 2. Ein Beispiel:

```
<html><head><title>orphans</title>
<style type="text/css">
  p,blockquote { orphans:3; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<h1>Zu dieser Seite</h1>
```

```

<p>Auf dieser Seite gibt es Abs&auml;tze.</p>
<blockquote>und Zitate</blockquote>
<p>Und alle haben gemeinsam: keiner wird ungl&uuml;cklich getrennt!</p>

</body></html>

```

### **CSS 2.0** widows (allein stehende Zeilen am Seitenanfang)

Sie können verhindern, dass Seitenumbrüche innerhalb von Fließtext den Text so unglücklich trennen, dass eine einzelne Zeile auf der neuen Seite steht und alle anderen Absatzzeilen auf der Seite vor dem Seitenumbruch bleiben (DTP-Fachjargon hierzulande: *Hurenkinder*). Mit `widow` bestimmen Sie, wie viele Zeilen eines längeren Fließtextabsatzes auf der Seite nach dem Umbruch auf der neuen Seite mindestens stehen sollen. Erlaubt ist eine Zahl für die Anzahl Zeilen. Voreinstellung ist 2. Ein Beispiel:

```

<html><head><title>widows</title>
<style type="text/css">
  p,blockquote { widows:3; }
</style>
</head><body bgcolor="#FFFFFF" text="#000000">

<h1>Zu dieser Seite</h1>
<p>Auf dieser Seite gibt es Abs&auml;tze.</p>
<blockquote>und Zitate</blockquote>
<p>Und alle haben gemeinsam: keiner wird ungl&uuml;cklich getrennt!</p>

</body></html>

```

### **CSS 2.0** Benannte Seiten

Für spezielle Seitenformate können Sie einzelne, benannte Seitentypen definieren. Ein Beispiel:

```

<style type="text/css">
  @page querformat { size:landscape; }
  div.uebersicht { page:querformat; /* andere CSS-Angaben */ }
</style>

```

Indem Sie hinter `@page` einen selbst vergebenen Namen notieren – im obigen Beispiel ist es der Name `querformat` –, können Sie ein spezielles Seitenlayout erstellen. Innerhalb der geschweiften Klammern können Sie die üblichen Angaben zum Seitenlayout notieren. Benannte Seiten werden jedoch nicht automatisch verwendet, sondern nur dann, wenn ein Element vorkommt, für das eine CSS-Eigenschaft `page`: notiert ist. Im Beispiel wird für `div`-Elemente mit dem Klassennamen `uebersicht` bestimmt, dass solche Elemente den Seitentyp `querformat` erzwingen. Wenn nun also im HTML-Quelltext steht:

```
<div class="uebersicht">...</div>
```

dann wird für die Darstellung dieses Elements der Seitentyp `querformat` erzwungen.

## 16.13 Sprachausgabe

### Allgemeines zur Sprachausgabe

Dank dieser CSS-Eigenschaften, die zur Version 2.0 des CSS-Sprachstandards gehören, wird es möglich, HTML-formatierten Text für die Sprachausgabe auf Audio-Systemen umzusetzen. Die CSS-Eigenschaften zur Sound-Kontrolle für Sprachausgabe erlauben es, Sprecherstimmen auf vielfältige Weise zu beeinflussen und den Einsatz von begleitenden Geräuschen oder Musikpassagen zu kontrollieren.

Die hier vorgestellten Möglichkeiten sind jedoch nicht nur für Sehbehinderte eine Hoffnung. Sie sind auch interessant für schöngestimmte Autoren, die Texte nicht nur schreiben, sondern auch für den Zweck des Vorlesens exakt formatieren wollen. Vor allem als Autorenanweisungen für Hör- und Bühnenstücke können diese Angaben sehr interessant sein.

Die computertechnische Umsetzung einer solchen akustischen Feinsteuerung von synthetischen Stimmen ist jedoch alles andere als trivial. »Ich hör ja gar nix« wird auch deshalb noch eine Weile lang das enttäuschende Erlebnis beim »Anhören« der Beispiele auf dieser Seite bleiben, denn die heute verbreiteten Browser sind rein bildschirmorientiert und besitzen keine Programm-Module für Sprachausgabe. Zwar gibt es mittlerweile Software zur Sprachausgabe von Web-Seiten, doch auch bei solchen Produkten ist man in der Regel noch nicht so weit, die hier vorgestellten CSS-Eigenschaften zu unterstützen.

Ein Problem bei der Sprachausgabe ist auch die Vielzahl der Landessprachen. Eine synthetische Stimme, die auf »Englisch« getrimmt ist, wird sich bei der Ausgabe eines deutschen Textes schauerlich anhören. Es ist deshalb auf jeden Fall empfehlenswert, in Dateien, die CSS-Eigenschaften für die Sprachausgabe enthalten, die verwendete Landessprache auszuweisen. Dies kann beispielsweise mit Hilfe des Universalattributs `lang=` (Kapitel 13.1) in einzelnen HTML-Elementen oder zentral im `body`-Element geschehen.

#### **CSS 2.0** **volume (Lautstärke)**

Sie können für ein HTML-Element oder einen Bereich angeben, in welcher Lautstärke er ausgegeben werden soll. Die Lautstärkeangabe ist eine relative Größe in Bezug auf die Grundlautstärke, die der Anwender eingestellt hat. Ein Beispiel:

```
<html><head><title>volume</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="volume:silent">Text wird mit der Lautstärke <b>silent</b>
gesprochen</p>
<p style="volume:x-loud">Text wird mit der Lautstärke <b>x-loud</b>
gesprochen</p>
<p style="volume:20">Text wird mit der Lautstärke <b>20</b> gesprochen</p>
<p style="volume:100">Text wird mit der Lautstärke <b>100</b> gesprochen</p>

</body></html>
```

Mit `volume:` können Sie die relative Lautstärke bestimmen. Erlaubt sind Werte zwischen 0 (sehr leise) und 100 (sehr laut) oder einer der folgenden Angaben:

| CSS-Angabe | Wirkung                                                   |
|------------|-----------------------------------------------------------|
| silent     | Ton aus, jedoch eine Sprechpause in der Länge der Passage |
| x-soft     | sehr leise, entspricht Wert 0                             |
| soft       | leise, entspricht Wert 25                                 |
| medium     | mittel, entspricht Wert 50                                |
| loud       | laut, entspricht Wert 75                                  |
| x-loud     | laut, entspricht Wert 100                                 |

### CSS 2.0 **speak (Aussprache)**

Sie können bestimmen, wie ein Element oder ein Bereich ausgesprochen wird. Mit `speak`: beeinflussen Sie die Aussprache. Folgende Angaben sind erlaubt:

- `none` = Ton aus und keine Sprechpause (Element wird quasi übergangen)
- `normal` = normale, sprachabhängige Aussprache
- `spell-out` = buchstabierte Aussprache (sinnvoll z.B. für Abkürzungen)

Ein Beispiel:

```
<html><head><title>speak</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="speak:normal">Das Wort &quot;Buchstabieren&quot; wird jetzt
buchstabiert:</p>
<p style="speak:spell-out">Buchstabieren</p>

</body></html>
```

### CSS 2.0 **pause-before (Pause vor einem Element)**

Sie können eine Sprechpause vor einem Element erzwingen. Mit `pause-before`: können Sie eine vorangehende Sprechpause erzwingen. Erlaubt ist eine Zeitangabe in Sekunden (s) oder Millisekunden (ms). Alternativ dazu können Sie auch Prozentwerte angeben. Dann ist die Sprechpause abhängig von der (ebenfalls einstellbaren) Sprechgeschwindigkeit. Ein Beispiel:

```
<html><head><title>pause-before</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<h1>Text mit Pausen</h1>
<p style="pause-before:300ms">Text mit 300ms Pause davor</p>
<p style="pause-before:600ms">Text mit 600ms Pause davor</p>
<p style="pause-before:1s">Text mit 1s Pause davor</p>
<p style="pause-before:2s">Text mit 2s Pause davor</p>

</body></html>
```



**CSS 2.0** *pause-after (Pause nach einem Element)*

Sie können mit `pause-after`: eine Sprechpause nach einem Element erzwingen. Erlaubt ist eine Zeitangabe in Sekunden (s) oder Millisekunden (ms). Alternativ dazu können Sie auch Prozentwerte angeben. Dann ist die Sprechpause abhängig von der (ebenfalls einstellbaren) Sprechgeschwindigkeit. Ein Beispiel:

```
<html><head><title>pause-after</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<h1>Text mit Pausen</h1>
<p style="pause-after:200ms">Text mit 200ms Pause danach</p>
<p style="pause-after:400ms">Text mit 400ms Pause danach</p>
<p style="pause-after:1s">Text mit 1s Pause danach</p>
<p style="pause-after:3s">Text mit 3s Pause danach</p>
<p>Abschließender Text</p>

</body></html>
```

**CSS 2.0** *pause (Pause vor/nach einem Element)*

Mit `pause`: können Sie ein Element durch Sprechpausen davor und danach abgrenzen. Wenn Sie einen Wert angeben, bedeutet dieser Wert sowohl die Sprechpause vor dem Element als auch nach dem Element. Sie können auch zwei Werte angeben, durch ein Leerzeichen getrennt. Dann bedeutet der erste Wert die Sprechpause vor dem Element und der zweite die Sprechpause danach. Erlaubt sind Zeitangaben in Sekunden (s) oder Millisekunden (ms). Alternativ dazu können Sie auch Prozentwerte angeben. Dann sind die Sprechpausen abhängig von der (ebenfalls einstellbaren) Sprechgeschwindigkeit. Ein Beispiel:

```
<html><head><title>pause</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<h1>Text mit Pausen</h1>
<p style="pause:500ms">Text mit 500ms Pause davor und danach</p>
<p style="pause:1300ms">Text mit 1300ms Pause davor und danach</p>
<p style="pause:1s 2s">Text mit 1s Pause davor und 2s Pause danach</p>
<p>Abschließender Text</p>

</body></html>
```

**CSS 2.0** *cue-before (Erkennungssignal vor einem Element)*

Mit `cue-before:url([Dateiname])` können Sie ein Erkennungssignal voranschicken. Wenn sich die Datei in einem anderen als dem aktuellen Verzeichnis befindet, müssen Sie als [Dateiname] relative Pfadangaben oder absolute URIs verwenden. Als Sound-Dateiformate kommen WAV, AU oder AI in Frage. Ein Beispiel:

```
<html><head><title>cue-before</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">
```

```
<p style="cue-before:url(cue.wav)">Freuen Sie sich!</p>
</body></html>
```

### **CSS 2.0** *cue-after (Erkennungssignal nach einem Element)*

Mit `cue-after:url([Dateiname])` können Sie ein Erkennungssignal an ein Element anschließen. Wenn sich die Datei in einem anderen als dem aktuellen Verzeichnis befindet, müssen Sie als `[Dateiname]` relative Pfadangaben oder absolute URIs verwenden. Als Sound-Dateiformate kommen WAV, AU oder AI in Frage. Ein Beispiel:

```
<html><head><title>cue-after</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="cue-after:url(cue.wav)">Und jetzt folgt ein:</p>

</body></html>
```

### **CSS 2.0** *cue (Erkennungssignal vor/nach einem Element)*

Mit `cue:url([Dateiname])` können Sie ein Element in zwei gleiche Erkennungssignale einschließen. Wenn sich die Datei in einem anderen als dem aktuellen Verzeichnis befindet, müssen Sie als `[Dateiname]` relative Pfadangaben oder absolute URIs verwenden. Als Sound-Dateiformate kommen WAV, AU oder AI in Frage. Ein Beispiel:

```
<html><head><title>cue</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="cue:url(cue.wav)">oh oh, so ein oh oh!</p>

</body></html>
```

### **CSS 2.0** *play-during (Hintergrund-Sound)*

Sie können für ein Element oder einen Bereich einen Hintergrund-Sound definieren, der so lange wiederholt wird, bis die Ausgabe des Elements oder Bereichs abgeschlossen ist. Das kann etwa eine Geräuschkulisse oder eine Melodie sein. Ein Beispiel:

```
<html><head><title>play-druring</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="play-during:url(play_during.mid) mix">W&uuml;hrend dieser Text gesprochen
wird, ert&ouml;nt im Hintergrund ein flotter Ragtime als MIDI-Datei. Damit Sie in
den Genu&szlig; des gesamten St&uuml;cks kommen, m&uuml;szlig;te ich Ihnen jetzt
eine lange Geschichte erz&uuml;hlen. Aber ich denke, Sie gewinnen auch ohne eine
solche Geschichte einen Eindruck davon, wie Hintergrundmusik w&uuml;hrend der
Sprachausgabe eines Textes wirkt. Vielen Dank f&uuml;r Ihre Aufmerksamkeit!</p>

</body></html>
```

Mit `play-during:url([Datei])` können Sie einen Hintergrund-Sound definieren. Damit ein echter Hintergrundeffekt zustande kommt, müssen Sie außerdem, durch ein Leerzeichen getrennt, das Wort `mix` dahinter notieren. Wenn Sie dieses Wort weglassen, übertönt der Hintergrund-Sound die Ausgabe des Inhalts. Wenn sich die Sound-Datei in einem anderen als dem aktuellen Verzeichnis befindet, müssen Sie als `[Dateiname]` relative Pfadangaben oder absolute URIs verwenden.

Hinter der Angabe der Datei können Sie noch eine der folgenden Angaben notieren:

- `mix` = Falls das Elternelement ebenfalls Hintergrund-Sound hat, werden beide Sounds vermischt.
- `repeat` = Wenn die Sound-Datei kürzer ist als es dauert, den Text des Elements zu sprechen, wird der Abspielvorgang so oft wiederholt, bis der Text zu Ende gesprochen ist.
- `auto` = Falls das Elternelement ebenfalls Hintergrund-Sound hat, wird dieser nicht neu abgespielt, sondern spielt weiter ab.

Anstelle einer Sound-Datei können Sie auch einfach nur `none` angeben. Das ist sinnvoll, wenn das Elternelement Hintergrund-Sound abspielt. Für die Sprachausgabe des aktuellen Elements können Sie die Ausgabe dann unterbrechen.



### **azimuth (Raum-Effekt links/rechts/vorne/hinten)**

Wenn das Sprachausgabesystem räumliches Hören (mindestens stereophon, möglichst Surround-Sound) unterstützt, können Sie die Herkunftsrichtung der Sprechstimme bestimmen.

Ein Beispiel:

```
<html><head><title>azimuth</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="azimuth:left-side">Diese Stimme hier sollte von links kommen.</p>
<p style="azimuth:left-side behind">Diese Stimme hier sollte von links hinten
kommen.</p>
<p style="azimuth:center">Diese Stimme hier sollte ganz aus der Mitte kommen.</p>
<p style="azimuth:right-side behind">Diese Stimme hier sollte von rechts hinten
kommen.</p>
<p style="azimuth:right-side">Diese Stimme hier sollte von rechts kommen.</p>

</body></html>
```

Mit `azimuth:` können Sie die räumliche Herkunftsrichtung der Sprechstimme für ein Element oder einen Bereich bestimmen. Sie können dazu entweder eine Gradangabe für die gedachte Richtung notieren oder ein erlaubtes Schlüsselwort. Gradangaben haben Werte zwischen 0 und 360 und müssen mit der Einheitenbezeichnung `deg` (*degree = Grad*) abgeschlossen werden, so wie im zweiten der obigen Beispiele. Folgende Schlüsselwörter sind erlaubt:

| CSS-Angabe          | Wirkung                                      |
|---------------------|----------------------------------------------|
| left-side           | von ganz links (entspricht 270deg)           |
| left-side behind    | von ganz links hinten (entspricht 270deg)    |
| far-left            | von weiter links (entspricht 300deg)         |
| far-left behind     | von weiter links hinten (entspricht 240deg)  |
| left                | von links (entspricht 320deg)                |
| left behind         | von links hinten (entspricht 220deg)         |
| center-left         | von Mitte links (entspricht 340deg)          |
| center-left behind  | von Mitte links hinten (entspricht 200deg)   |
| center              | von der Mitte (entspricht 0deg)              |
| center behind       | von Mitte hinten (entspricht 180deg)         |
| center-right        | von Mitte rechts (entspricht 20deg)          |
| center-right behind | von Mitte rechts hinten (entspricht 160deg)  |
| right               | von rechts (entspricht 40deg)                |
| right behind        | von rechts hinten (entspricht 140deg)        |
| far-right           | von weiter rechts (entspricht 60deg)         |
| far-right behind    | von weiter rechts hinten (entspricht 120deg) |
| right-side          | von ganz rechts (entspricht 90deg)           |
| right behind        | von ganz rechts hinten (entspricht 90deg)    |

Wenn Sie mit Gradangaben arbeiten, können Sie die genannten Angaben als Richtwerte für die exakt gewünschte Richtung verwenden. Anstelle des Schlüsselwortes `behind` können Sie auch die Schlüsselwörter `leftwards` und `rightwards` benutzen. Mit `leftwards` richten Sie die Stimmherkunft weiter links aus (20deg weniger), mit `rightwards` weiter rechts (20deg mehr).

### **CSS 2.0** *elevation (Raum-Effekt oben/unten)*

Wenn das Sprachausgabesystem räumliches Hören (möglichst Surround-Sound) unterstützt, können Sie die Herkunftsrichtung der Sprechstimme zwischen oben und unten bestimmen. Ein Beispiel:

```
<html><head><title>elevation</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="elevation:below">Diese Stimme sollten von unten kommen.</p>
<p style="elevation:higher">Diese Stimme sollte von weiter oben kommen.</p>

</body></html>
```

Mit `elevation`: können Sie die vertikale Herkunftsrichtung der Sprechstimme bestimmen. Sie können dazu entweder eine Gradangabe für die gedachte Richtung notieren oder ein erlaubtes Schlüsselwort. Gradangaben haben Werte zwischen -90 (von ganz unten) und 90 (von ganz oben) und müssen mit der Einheitenbezeichnung `deg` (*degree = Grad*) abgeschlossen werden, so wie im zweiten der obigen Beispiele. Folgende Schlüsselwörter sind erlaubt:

| CSS-Angabe | Wirkung          |
|------------|------------------|
| below      | von unten        |
| lower      | von weiter unten |
| level      | von der Mitte    |
| above      | von oben         |
| higher     | von weiter oben  |

### **CSS 2.0** *speech-rate (Sprechgeschwindigkeit)*

Sie können die Schnelligkeit der gesprochenen Wiedergabe beeinflussen. Ein Beispiel:

```
<html><head><title>speech-rate</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="speech-rate:fast">Das ist ein ziemlich schnell gesprochener Text.
Hoffentlich verstehen Sie alles, was da gesagt wird, auch wenn es nicht von Belang
ist.</p>
<p style="speech-rate:x-slow">Das ist ein gaaaannnz langsam gesprochener Text. Gute
Nacht!</p>

</body></html>
```

Mit `speech-rate`: können Sie die Sprechgeschwindigkeit bestimmen. Erlaubt sind folgende Angaben:

| CSS-Angabe | Wirkung                    |
|------------|----------------------------|
| x-slow     | ganz langsam               |
| slow       | langsam                    |
| slower     | etwas langsamer als normal |
| medium     | normal                     |
| faster     | etwas schneller als normal |
| fast       | schnell                    |
| x-fast     | ganz schnell               |

Anstelle der Schlüsselwörter können Sie auch numerische Werte angeben. Ein numerischer Wert bedeutet die mittlere Anzahl Wörter pro Minute. Probieren Sie als Anhaltspunkt Werte um 100.

### **CSS 2.0** *voice-family (Sprechertyp)*

Sie können die Art der Stimme beeinflussen. Dies entspricht etwa dem Beeinflussen von Schriftarten bei visueller Darstellung. Ob die Angabe interpretiert werden kann, hängt neben der interpretierenden Software von den Hardwarefähigkeiten des Sprachausgabesystems ab. Ein Beispiel:

```
<html><head><title>voice-family</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="voice-family:female">Ich bin die universelle Frau. Mein Herz ist meine
Familie.</p>
<p style="voice-family:male">Ich bin der universelle Mann. Mein Herz ist meine
Frau.</p>

</body></html>
```

Mit `voice-family`: können Sie die Art der Sprechstimme festlegen. Folgende generische Angaben werden vom W3-Konsortium z.B. vorgeschlagen:

- `male` = Männerstimme
- `female` = Frauenstimme
- `child` = Kinderstimme

Daneben können Sie auch spezifische Angaben notieren, etwa `MeineEhefrau`. Die entsprechende Stimme muss dem Sprachausgabesystem in digitaler Form bekannt sein.

### ***pitch (Sprehhöhe)***

Mit `pitch`: können Sie die Sprehhöhe festlegen. Erlaubt ist entweder die Angabe der mittleren Sprehhöhe als Frequenz (in Hz für Hertz oder kHz für Kilohertz) oder eine der folgenden Angaben:

| CSS-Angabe          | Wirkung   |
|---------------------|-----------|
| <code>x-low</code>  | ganz tief |
| <code>low</code>    | tief      |
| <code>medium</code> | normal    |
| <code>high</code>   | hoch      |
| <code>x-high</code> | ganz hoch |

Ein Beispiel:

```
<html><head><title>pitch</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="pitch:x-low">Diese Stimme spricht so tief wie möglich zu Ihnen.
Lassen Sie sich trotzdem nicht einsch&uuml;chtern.</p>
<p style="pitch:x-high">Diese Stimme spricht so hoch wie möglich zu Ihnen.
Schnatterdischnatterdischnatterdischnatt.</p>

</body></html>
```

**CSS 2.0** *pitch-range (Sprechweise)*

Mit `pitch-range`: können Sie die Sprechweise bestimmen. Normale Sprechweise entspricht einem Wert von 50. Ein tieferer Wert bedeutet eine monotone, modulationsarme Stimme, ein hoher Wert eine angeregte, modulationsreiche Stimme. Ein Beispiel:

```
<html><head><title>pitch-range</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="pitch-range:20">Dies ist ein ziemlich &ouml;der Text, deshalb wird der
mit ziemlich monotoner Stimme gesprochen.</p>
<p style="pitch-range:80">Obwohl auch dies ein ziemlich &ouml;der Text ist, wird er
dennoch mit sehr angeregter Stimme vorgetragen.</p>

</body></html>
```

**CSS 2.0** *stress (Sprechhast)*

Sie können die Emphatik der Stimme beeinflussen. Mit `stress`: können Sie festlegen, wie hastig die Stimme ist. Ein Wert von 50 ist normal. Ein höherer Wert bedeutet eine hastige, aufgeregte Stimme, ein niedriger Wert eine in sich ruhende, selbstsichere Stimme. Ein Beispiel:

```
<html><head><title>stress</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="stress:10">Ich habe alle Zeit der Welt. Deshalb bin ich auch ganz
gelassen und spreche nur mit einem Stress-Wert von 10.</p>
<p style="stress:90">Immerzu ist alles gleich vorbei. Nie kommt man zu etwas. Alles
ist grausam. Deshalb rede ich auch so hastig. Ein Stress-Wert von 90 wurde mir
vorgegeben, stellen Sie sich das mal vor!</p>

</body></html>
```

**CSS 2.0** *richness (Stimmgewalt)*

Sie können die Wichtigkeit der Stimme beeinflussen. Mit `richness`: können Sie die Stimmgewalt festlegen. Normalwert ist 50. Ein höherer Wert bedeutet eine gewaltige, beherrschende Stimme, ein niedriger Wert eine dünne, unsichere Stimme. Ein Beispiel:

```
<html><head><title>richness</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="richness:85">Ich bin reich. Ich bin sch&ouml;n.
Deshalb rede ich laut und klar, mit einer Stimmgewalt von 85.</p>
<p style="richness:15">Ich bin arm. Ich bin h&auml;sslich. Deshalb hab ich
auch so eine d&uuml;nne Stimme. Ihre Gewalt hat nur einen Wert von 15.</p>

</body></html>
```

**CSS 2.0**  *speak-punctuation (Satzzeichen explizit mit ausgeben)*

Sie können bestimmen, dass bei einem Textbereich alle darin vorkommenden Satzzeichen wie Punkt, Komma, Ausrufezeichen usw. explizit genannt werden, also z.B. »Dies ist ein Satz – Punkt«. Ein Beispiel:

```
<html><head><title>speak-punctuation</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p style="speak-punctuation:code">H&ouml;ren Sie mal! - Oder sind Sie
schwerh&ouml;righ? Hier - und nur hier gibt's was zus&auml;tzlich auf die Ohren:
n&auml;mlich exklusiv ausgesuchte, explizit ausgesprochene Satzzeichen.</p>

</body></html>
```

Mit `speak-punctuation`: haben Sie Einfluss auf die Nennung von Satzzeichen. Folgende Angaben sind erlaubt:

- `code` = Satzzeichen mit aussprechen
- `none` = Satzzeichen nicht mit aussprechen (Normaleinstellung)

**CSS 2.0**  *speak-numeral (Konvention für Zahlenausgabe)*

Sie können bestimmen, in welcher Form größere Zahlen, z.B. 4256, ausgesprochen werden. Ein Beispiel:

```
<html><head><title>speak-numeral</title>
</head><body lang="de" bgcolor="#FFFFFF" text="#000000">

<p>Rosi's Telefonnummer lautet <b style="speak-numeral:digits">32168</b>.</p>
<p>Mit ihrem Gewerbe hat Rosi schon <b style="speak-numeral:continous">32168</b>
Dollar verdient.</p>

</body></html>
```

Mit `speak-numeral`: können Sie festlegen, wie Zahlenfolgen ausgesprochen werden. Erlaubt sind folgende Angaben:

- `digits` = ziffernweise, 456 also etwa als »vier fünf-sechs«
- `continous` = als Zahl, 456 also etwa »vierhundertsechsfünfzig«

## 16.14 Anzeigefenster

### Allgemeines zum Anzeigefenster

Zum Anzeigefenster gibt es zunächst einmal zu sagen, dass es Teil der Arbeitsumgebung des Anwenders ist und deshalb kein Spiel- und Tummelplatz für Autoren von Web-Seiten. Die einzige Eigenschaft des Anzeigefensters, auf die das W3-Konsortium in der CSS-Spezifikation den Zugriff ermöglicht, ist das Aussehen des Mauszeigers. Das ist insofern sinnvoll, als es zur Funktionalität von Web-Seiten unterstützend beitragen kann.



Über den Sinn der CSS-Eigenschaften, die Microsoft in den Internet Explorer seit Version 5.5 eingebaut hat, kann man dagegen geteilter Meinung sein. Im Rahmen stimmiger Web-Designs kann es sinnvoll sein, dass auch die Scroll-Leisten des Browser-Fensters oder die von mehrzeiligen Formulareingabefeldern zu den übrigen Farben passen. Andererseits wecken solche Möglichkeiten wieder neue Begehrlichkeiten in jenen Leuten, die über ihre Web-Seiten am liebsten den Computer des Anwenders ein- und ausschalten würden. Denn natürlich wollen die als Nächstes auch die Menü- und Werkzeugleiste des Browsers gestalten können.

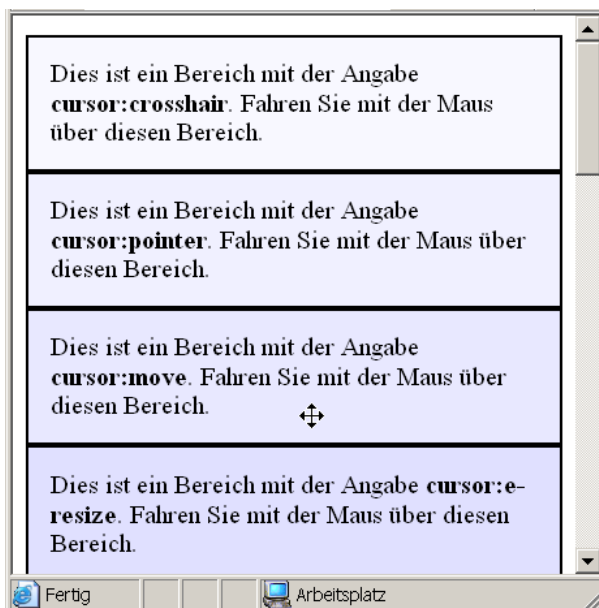
### 16.14.1 cursor (Mauszeiger)

Sie können für ein HTML-Element einen Cursor definieren. Wenn der Anwender mit der Maus über die Fläche fährt, die das Element einnimmt, nimmt der Mauscursor die angegebene Gestalt an. Da Mauszeiger immer mit einer Bedeutung versehen sein sollten, ist es in der Praxis sinnvoll, solche Elemente etwa mit Hilfe von JavaScript Event-Handlern an ein Script zu binden, das bestimmte Befehle ausführt. Ein Beispiel:

```
<html><head><title>cursor</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<div style="background-color:#F8F8FF; padding:10pt; border:thin solid black;
width:100%; cursor:crosshair;">
<p>Dies ist ein Bereich mit der Angabe <b>cursor:crosshair</b>.
Fahren Sie mit der Maus über diesen Bereich.</p></div>

<!-- usw. -->
</body></html>
```



16.51: Die CSS-Angabe für den Bereich beeinflusst den Mauszeiger.

Mit `cursor`: können Sie das Aussehen des Mauszeiger für den Fall definieren, dass der Anwender mit der Maus über den entsprechenden Bereich fährt. Folgende Angaben sind erlaubt:

| CSS-Angabe                | Wirkung                                                                               |
|---------------------------|---------------------------------------------------------------------------------------|
| <code>auto</code>         | automatischer Cursor (Normaleinstellung)                                              |
| <code>default</code>      | plattformunabhängiger Standard-Cursor                                                 |
| <code>crosshair</code>    | Cursor in Form eines einfachen Fadenkreuzes                                           |
| <code>pointer</code>      | Cursor in Form eines Zeigers                                                          |
| <code>move</code>         | Cursor in Form eines Kreuzes, das die Fähigkeit zum Bewegen des Elements signalisiert |
| <code>n-resize</code>     | Cursor in Form eines Pfeils, der nach oben zeigt (n: Norden)                          |
| <code>ne-resize</code>    | Cursor in Form eines Pfeils, der nach rechts oben zeigt (ne: Nordost)                 |
| <code>e-resize</code>     | Cursor in Form eines Pfeils, der nach rechts zeigt (e: Osten)                         |
| <code>se-resize</code>    | Cursor in Form eines Pfeils, der nach rechts unten zeigt (se: Südost)                 |
| <code>s-resize</code>     | Cursor in Form eines Pfeils, der nach unten zeigt (s: Süden)                          |
| <code>sw-resize</code>    | Cursor in Form eines Pfeils, der nach links unten zeigt (sw: Südwest)                 |
| <code>w-resize</code>     | Cursor in Form eines Pfeils, der nach links zeigt (w: Westen)                         |
| <code>nw-resize</code>    | Cursor in Form eines Pfeils, der nach links oben zeigt (nw: Nordwest)                 |
| <code>text</code>         | Cursor in einer Form, die normalen Text symbolisiert                                  |
| <code>wait</code>         | Cursor in Form eines Symbols, das einen Wartezustand signalisiert                     |
| <code>help</code>         | Cursor in Form eines Symbols, das Hilfe zu dem Element signalisiert                   |
| <code>url([Datei])</code> | Beliebiger Cursor, [Datei] sollte eine GIF- oder JPG-Grafik sein                      |

**Beachten Sie:** Cursor-Symbole sollten Sie nicht zweckentfremden. Wenn Sie beispielsweise `cursor:move` für ein Element benutzen, sollte dieses Element auch tatsächlich beweglich sein (etwa mit Hilfe von Dynamischem HTML), und wenn Sie `cursor:help` benutzen, erwartet der Anwender beim Anklicken des Bereichs, dass am Bildschirm ein Fenster oder ein Bereich mit einem Hilfetext erscheint. Netscape 6.x und Internet Explorer 5.x interpretieren die Angabe `url([Datei])` nicht.



### **scrollbar (Microsoft-Eigenschaften für Scroll-Leiste**

Die hier beschriebenen Eigenschaften lassen sich auf HTML-Elemente wie `body` oder `textarea` anwenden. Bei `body` legen Sie die Farben der Fenster-Scroll-Leisten fest, bei `textarea` die Farben der Scroll-Leisten des Eingabebereichs. Ein Beispiel:

```
<html><head><title>scrollbars</title>
<style type="text/css">
body, textarea {
  background-color:#8C8CC6;
  color:#FFFFFF;
  font-family:"Andale Mono",Arial,sans-serif;
  scrollbar-base-color:#8C8CC6;
  scrollbar-3dlight-color:#FFFFFF;
  scrollbar-arrow-color:#FFFFFF;
  scrollbar-darkshadow-color:#000000;
```

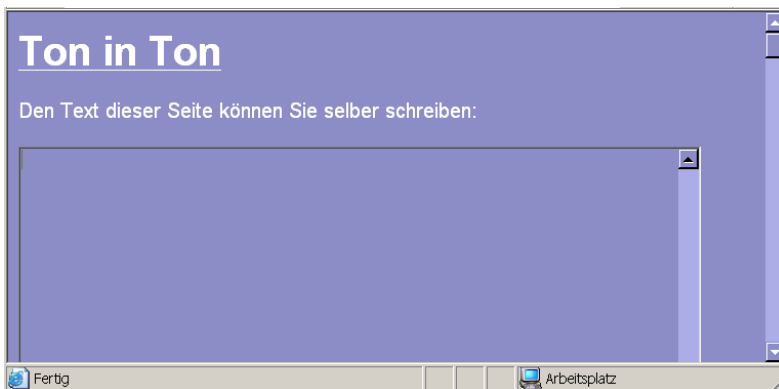
```

scrollbar-face-color:#8C8CC6;
scrollbar-highlight-color:#FFFFFF;
scrollbar-shadow-color:#000000;
scrollbar-track-color:#ACACE6;
}
</style>
</head><body>
<h1 style="text-decoration:underline">Ton in Ton</h1>

<form name="Seite">
<p>Den Text dieser Seite können Sie selber schreiben:</p>
<textarea name="Text" cols="80" rows="200"></textarea>
</form>

</body></html>

```



16.52: Die einheitliche Farbgestaltung umfasst auch die Rollbalken in Eingabefeldern und am Browser-Rand.

Mit den folgenden Eigenschaften können Sie die Einzelteile von Scroll-Leisten bestimmen. Bei all diesen Eigenschaften ist eine Farbangabe möglich.

| CSS-Angabe                 | Wirkung                                 |
|----------------------------|-----------------------------------------|
| scrollbar-base-color       | Basisfarbe der Scroll-Leiste            |
| scrollbar-3dlight-color    | Farbe für 3-D-Effekte                   |
| scrollbar-arrow-color      | Farbe für Verschiebepfeile              |
| scrollbar-darkshadow-color | Farbe für Schatten                      |
| scrollbar-face-color       | Farbe für Oberfläche                    |
| scrollbar-highlight-color  | Farbe für oberen und linken Rand        |
| scrollbar-shadow-color     | Farbe für unteren und rechten Rand      |
| scrollbar-track-color      | Farbe für frei bleibenden Verschiebeweg |

## 16.15 Filter (nur Microsoft)

### Allgemeines zu Filtern

Die hier vorgestellten CSS-Eigenschaften kommen aus dem Bereich der Bildbearbeitung. Die Filter berechnen Effekte, die sich auf geeignete Objekte anwenden lassen. Durch diese Filter werden zwar sehr interessante optische Effekte möglich. Auch kunstvolle Textgestaltung ist damit möglich, was manche Grafik erspart. Dennoch sind all die CSS-Eigenschaften für Filter kein CSS-Standard und ein proprietäres Leistungsmerkmal des Internet Explorers. Andere Browser interpretieren diese Eigenschaften nicht. Sie können für einzelne Elemente Filter angeben, die das Element dynamisch verändern. Das Prinzip ist immer das gleiche. Ein Beispiel:

```


```

Mit `filter:` können Sie einen Effekt-Filter bestimmen. Hinter dem Doppelpunkt folgt der Aufruf eines Filters. Einige Filter erwarten Parameter, andere nicht. Die Parameter und Ihre Wertangaben zu den Filtern notieren Sie in Klammern unmittelbar hinter dem Filternamen, so wie im obigen Beispiel (`strength=50`) als Parameter an den Filter `Blur()` übergeben wird. Wenn ein Filter keine Parameter erwartet, notieren Sie trotzdem die Klammern, lassen den Inhalt dazwischen aber leer.

Sie können auch mehrere Filter kombinieren. Im zweiten der obigen Beispiele werden etwa die beiden Filter `Blur()` und `FlipH()` kombiniert. Dazu dürfen Sie nur einmal das Schlüsselwort `filter:` notieren. Zwischen diesem Schlüsselwort und dem nächsten Semikolon (;) zum Abschließen der Stylesheet-Angabe können Sie mehrere Filter notieren. Trennen Sie die Filter durch ein Leerzeichen vor dem Namen des nächsten Filters, so wie im Beispiel vor `FlipH()`. Die Filter sind für die folgenden HTML-Elemente gedacht (bei anderen funktionieren sie nicht):

| HTML-Element | Wirkung                                                                                                                                                 |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| body         | Effekt betrifft die gesamte angezeigte Seite.                                                                                                           |
| button       | Effekt betrifft die Schaltfläche.                                                                                                                       |
| div          | Effekt betrifft den Bereich in Verbindung mit den Stylesheet-Angaben <code>width:</code> und/oder <code>height:</code> und/oder <code>position</code> . |
| img          | Effekt betrifft die referenzierte Grafik.                                                                                                               |
| input        | Effekt betrifft das Eingabefeld.                                                                                                                        |
| marquee      | Effekt betrifft den Lauftext.                                                                                                                           |
| span         | Effekt betrifft den Bereich in Verbindung mit den Stylesheet-Angaben <code>width:</code> und/oder <code>height:</code> und/oder <code>position</code> . |
| table        | Effekt betrifft die Tabelle.                                                                                                                            |
| td           | Effekt betrifft die Datenzeile einer Tabelle.                                                                                                           |
| textarea     | Effekt betrifft das mehrzeilige Eingabefeld.                                                                                                            |
| tfoot        | Effekt betrifft den Fußbereich einer Tabelle.                                                                                                           |
| th           | Effekt betrifft die Kopfzeile einer Tabelle.                                                                                                            |
| thead        | Effekt betrifft den Kopfbereich einer Tabelle.                                                                                                          |
| tr           | Effekt betrifft die Zeile einer Tabelle.                                                                                                                |

**Beachten Sie:** Es gibt noch drei weitere Microsoft-Filter, die jedoch Script-Sprachen-Unterstützung benötigen. Überhaupt werden diese Filter erst so richtig interessant, wenn sie dynamisch verändert werden. Lesen Sie dazu im Kapitel 21 über *Dynamisches HTML* den Abschnitt 21.6 *Dynamische Filter (nur Microsoft)*.

Auf dieser Seite werden die Filter beschrieben, die mit dem Internet Explorer 4.0 eingeführt wurden. Mit der Version 5.5 seines Browsers hat Microsoft wieder eine neue, erweiterte Syntax eingeführt. An die Stelle des Syntax-Schemas

```
filter:filtername(Eigenschaften)
```

tritt das neue Syntax-Schema:

```
filter:progid:DXImageTransform.Microsoft.filtername(Eigenschaften)
```



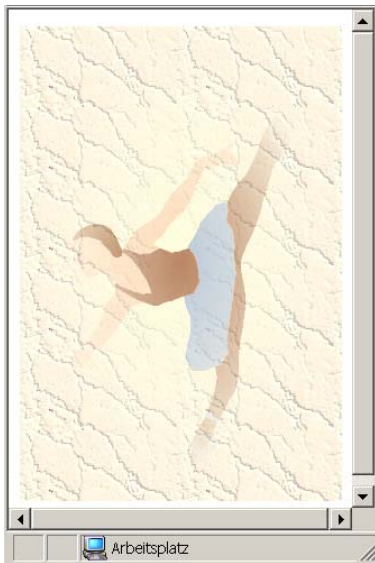
### **filter:Alpha() (Vordergrund/Hintergrund verschmelzen)**

Mit diesem Filter können Sie Vordergrundelemente mit Hintergrundelementen farblich verschmelzen. Der Filter erlaubt verschiedene Möglichkeiten, wie ein Vordergrundelement aus seinem Hintergrund »heraustreten« kann. Ein Beispiel:

```
<html><head><title>filter:Alpha()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<div style="background-image:url(background3.jpg); padding:20px; text-align:center">
</div>

</body></html>
```



16.53: Das Bild und der Bereichshintergrund gehen fließend ineinander über.

Mit `filter:Alpha()` können Sie Vordergrundelemente und Hintergrundelemente ineinander verschmelzen. Geeignet ist der Filter z.B. für Grafiken, die auf Hintergrundgrafiken erscheinen. Dieser Filter erwartet folgende Parameter:

- `opacity`= Deckgrad am Ursprung der Verschmelzung. Erlaubt sind Werte zwischen 0 und 100. Wert 0 bedeutet: vollkommen transparent (Hintergrund scheint voll durch), Wert 100 bedeutet: vollkommen abdeckend (Hintergrund scheint nicht durch).
- `finishopacity`= Deckgrad am Ende der Verschmelzung. Erlaubt sind Werte zwischen 0 und 100. Wert 0 bedeutet: vollkommen transparent (Hintergrund scheint voll durch), Wert 100 bedeutet: vollkommen abdeckend (Hintergrund scheint nicht durch).
- `style`= Art, in der der Filter wirkt. Erlaubt sind die Werte 0, 1, 2 und 3.  
 0 bedeutet: Hintergrund und Vordergrund werden farblich addiert. Das Vordergrundelement wird farblich entsprechend manipuliert. Keine Verschmelzung. Angaben zu den anderen Parametern sind in diesem Fall nicht erforderlich, d.h. `Alpha(style=0)` genügt.  
 1 bedeutet: linearer Verlauf von einem selbst definierbaren Anfangspunkt zu einem selbst definierbaren Endpunkt. Am Anfangspunkt gilt der Wert, der bei `opacity`= angegeben wird. Ab dem Endpunkt verliert der Wert für `opacity`= seinen Einfluss, und der Wert für `finishopacity`= dominiert. Den Anfangspunkt bestimmen Sie durch `startx`= und `starty`=, den Endpunkt durch `finishx`= und `finisly`=.  
 2 bedeutet: radialer (elliptischer) Verlauf von innen nach außen. Ganz innen gilt der angegebene Wert für `opacity`=, ganz außen der angegebene Wert für `finishopacity`=, In diesem Fall können Sie Angaben zu `startx`=, `starty`=, `finishx`= und `finisly`= weglassen, so wie im obigen Beispiel.  
 3 bedeutet: rechteckiger Verlauf von innen nach außen. Ganz innen gilt der angegebene Wert für `opacity`=, ganz außen der angegebene Wert für `finishopacity`=, In diesem Fall können Sie Angaben zu `startx`=, `starty`=, `finishx`= und `finisly`= weglassen, so wie im obigen Beispiel.
- `startx`= Anfangspunkt in Pixel horizontal vom linken Rand des Vordergrundelements. Nur in Verbindung mit `style=1` von Bedeutung.
- `starty`= Anfangspunkt in Pixel vertikal vom oberen Rand des Vordergrundelements. Nur in Verbindung mit `style=1` von Bedeutung.
- `finishx`= Endpunkt in Pixel horizontal vom linken Rand des Vordergrundelements. Nur in Verbindung mit `style=1` von Bedeutung.
- `finisly`= Endpunkt in Pixel vertikal vom oberen Rand des Vordergrundelements. Nur in Verbindung mit `style=1` von Bedeutung.



### **`filter:Blur()` (Verwisch-Effekt)**

Dieser Filter erlaubt es, Grafiken in HTML mit Verwisch-Effekt auszustatten. Ein Beispiel:

```
<html><head><title>filter:Blur()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

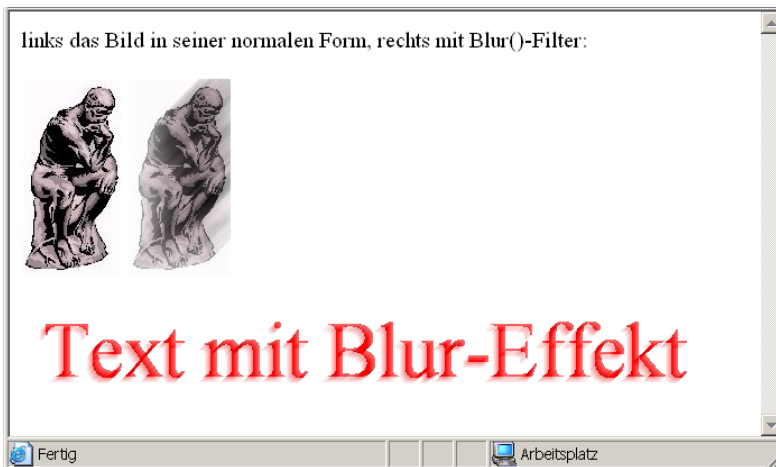
<p>links das Bild in seiner normalen Form, rechts mit Blur()-Filter:</p>
```

```



<div style="width:100%; padding:20px; font-size:42pt; color:red;
    filter:Blur(direction=235, strength=6)">Text mit Blur-Effekt</div>

</body></html>
```



16.54: Eine Grafik (rechts) und Text mit Verwisch-Effekt.

Mit `filter:Blur()` können Sie einen Verwisch-Effekt definieren. Geeignet ist der Filter für Grafiken. Dieser Filter erwartet folgende Parameter:

- `add=` Diesen Parameter brauchen Sie nur anzugeben, wenn Sie `add=0` notieren wollen. Dann erscheinen die Konturen der Grafik gar nicht mehr, und am Bildschirm erscheint nur noch die Verwisch-Spur. Der Effekt wird dadurch sehr stark, die eigentliche Grafik ist oft nicht mehr zu erkennen.
- `direction=` Mit diesem Parameter geben Sie die Richtung an, in die die Verwisch-Spur führt. Es wirkt dann so, als ob die Grafik aus dieser Richtung kommt. Erlaubt sind die Werte 0, 45, 90, 135, 180, 225, 270, 315:
  - 0 bedeutet: Die Verwisch-Spur zeigt nach oben.
  - 45 bedeutet: Die Verwisch-Spur zeigt nach oben rechts.
  - 90 bedeutet: Die Verwisch-Spur zeigt nach rechts.
  - 135 bedeutet: Die Verwisch-Spur zeigt nach unten rechts.
  - 180 bedeutet: Die Verwisch-Spur zeigt nach unten.
  - 225 bedeutet: Die Verwisch-Spur zeigt nach unten links.
  - 270 bedeutet: Die Verwisch-Spur zeigt nach links.
  - 315 bedeutet: Die Verwisch-Spur zeigt nach oben links.
- `strength=` Mit diesem Parameter bestimmen Sie, wie stark der Verwisch-Effekt sein soll. 0 bedeutet keinen Verwisch-Effekt, jeder höhere Wert die Verwisch-Spur in Pixeln. Beachten

Sie jedoch, dass die Grafik innerhalb ihrer normalen Bildgröße dargestellt wird. Verwisch-Spur von Konturen, die sehr nah am Bildrand sind, werden nur bis zum Bildrand gezogen.



### **filter:Chroma() (Transparenzfarbe)**

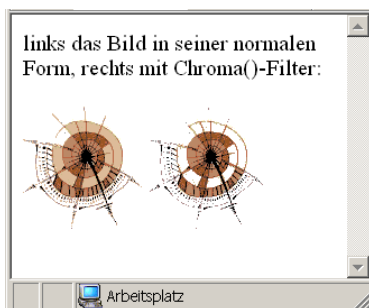
Mit diesem Filter können Sie bei Grafiken eine Farbe als transparent definieren. Anders als bei transparenten GIF-Grafiken (Kapitel 24.3.6) sind jedoch auch alle anderen Farben der Grafik betroffen. Mit `filter:Chroma()` können Sie eine Farbe in einer Grafik als transparent definieren. Der Filter erwartet folgenden Parameter: `color=` Eine Farbangabe. Ein Beispiel:

```
<html><head><title>filter:Chroma()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>links das Bild in seiner normalen Form, rechts mit Chroma()-Filter:</p>


</pre>

</body></html>
```



16.55: Im rechten Bild ist eine Farbe als transparent definiert.



### **filter:DropShadow() (Schattenwurf)**

Dieser Filter bewirkt einen Schatteneffekt für alle Konturen eines Elements. Bei Text bilden die Zeichen die Konturen, bei Grafiken sollten Sie nur solche für den Filter benutzen, die eindeutige Konturen haben, z.B. Cliparts. Ein Beispiel:

```
<html><head><title>filter:DropShadow()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<div style="width:100%; font-size:64pt; color:blue;
filter:DropShadow(color=#C0C0C0, offx=3, offy=3)">
Text mit Schattenwurf</div>

</body></html>
```





16.56: Eine Schrift mit dem DropShadow-Effekt.

Mit `filter:DropShadow()` können Sie einen Schattenwurf erzwingen. Der Filter erwartet folgende Parameter:

- `color`= Eine Farbe, die als Schatten definiert werden soll (siehe Farbangaben).
- `offx`= Anzahl für den Schatteneffekt horizontal in Pixeln. Positive Zahlen bewirken einen Schatteneffekt rechts, negative mit Minuszeichen davor einen Schatteneffekt links.
- `offy`= Anzahl für den Schatteneffekt vertikal in Pixeln. Positive Zahlen bewirken einen Schatteneffekt unten, negative mit Minuszeichen davor einen Schatteneffekt oben.
- `positive`= Diesen Parameter sollten Sie nur angeben, wenn Sie nur für die transparenten Pixel einer Grafik (siehe transparente GIF-Grafiken, Kapitel 24.3.6) einen Schatteneffekt erzwingen wollen. Dazu sollte dieser Parameter den Wert 1 erhalten.

**Beachten Sie:** Für verlaufende Schatten können Sie den Filter für `filter:Shadow()` verwenden.



#### ***filter:FlipH() (horizontal spiegeln)***

Mit `filter:FlipH()` bewirken Sie die horizontale Spiegelung. Geeignet ist der Filter für Grafiken in HTML. Dieser Filter erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>filter:FlipH()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Links die normale Grafik, rechts mit FlipH()-Filter:</p>




</body></html>
```



16.57: Der Filter spiegelt die Grafik auf der rechten Seite.

**filter:FlipV() (vertikal spiegeln)**

Mit `filter:FlipV()` bewirken Sie die vertikale Spiegelung. Geeignet ist der Filter für Grafiken in HTML. Dieser Filter erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>filter:FlipV()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Links die normale Grafik, rechts mit FlipV()-Filter:</p>




</body></html>
```



16.58: Der Filter stellt die Grafik auf den Kopf.

**filter:Glow() (glühender Rand)**

Dieser Filter bewirkt einen »Heiligenschein« um ein Text- oder Grafikobjekt. Ein Beispiel:

```
<html><head><title>filter:Glow()</title>
</head><body bgcolor="#FFFFFF" text="#000000">
```

```
<div style="width:100%; font-size:64pt; color:#9999CC;
    filter:Glow(color=#000000, strength=12)">Glühender Text</div>

</body></html>
```



16.59: Der Glüheffekt ist ganz nett, reicht aber nicht an die Fähigkeiten von Grafikprogrammen heran.

Mit `filter:Glow()` können Sie einen glühenden Rand definieren. Dieser Filter erwartet folgende Parameter:

- `color=` Eine Farbe, die als Glühfarbe definiert werden soll.
- `strength=` Stärke der Wirkung. Erlaubt sind Werte zwischen 1 und 255, praxisgerecht sind aber in den meisten Fällen nur Werte zwischen 1 und etwa 20.



#### ***filter:Gray() (Graustufen)***

Dieser Filter entfernt alle Farbinformationen aus einem Element oder einer Grafik und wandelt sie in Graustufen um. Mit `filter:Gray()` können Sie Graustufen erzwingen. Geeignet ist der Filter besonders für Grafiken. Dieser Filter erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>filter:Gray()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Links die normale Grafik, rechts mit Gray()-Filter:</p>




</body></html>
```



#### ***filter:Invert() (Invertieren)***

Mit `filter:Invert()` können Sie invertieren. Alle Farben werden bei der Anzeige in ihre Komplementärfarben verwandelt. Geeignet ist der Filter besonders für Grafiken. Dieser Filter erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>filter:Invert()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Links die normale Grafik, rechts mit Invert()-Filter:</p>
```

```



</body></html>
```



16.60: Der Filter `Invert()` wandelt alle Farben in ihre Komplementärfarben um.



#### **`filter:Mask()` (Transparenzmaske)**

Dieser Filter ist für transparente GIF-Grafiken gedacht. Er ersetzt alle transparenten Pixel der Grafik durch eine gewünschte Farbe und alle nichttransparenten Pixel der Grafik durch die Farbe, die in der Grafik als transparent definiert ist. Mit `filter:Mask()` können Sie für eine transparente GIF-Grafik eine Transparenzmaske definieren. Der Filter erwartet den Parameter `color=`, der die Farbe der Maske angibt. Ein Beispiel:

```
<html><head><title>filter:Mask()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Links die normale Grafik, rechts mit Mask()-Filter:</p>




</body></html>
```



16.61: Rechts sehen Sie alle Pixel, die im GIF-Bild als transparent gekennzeichnet sind.



### **filter:Shadow() (Schatten)**

Dieser Filter bewirkt einen Verlaufsschatten um die Konturen eines Elements. Ein Beispiel:

```
<html><head><title>filter:Shadow()/</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<div style="width:100%; font-size:48pt; color:magenta; filter:Shadow(color=#000000,
direction=45)">
Text mit Schatten</div>

</body></html>
```



16.62: Der Shadow-Filter macht seinem Namen alle Ehre.

Mit `filter:Shadow()` können Sie einen Schatten erzeugen. Dieser Filter erwartet folgende Parameter:

- `color=` Eine Schattenfarbe
- `direction=` Mit diesem Parameter geben Sie die Richtung an, in die der Schatteneffekt führt. Erlaubt sind die Werte 0, 45, 90, 135, 180, 225, 270, 315:  
 0 bedeutet: Der Schatteneffekt zeigt nach oben.  
 45 bedeutet: Der Schatteneffekt zeigt nach oben rechts.  
 90 bedeutet: Der Schatteneffekt zeigt nach rechts.  
 135 bedeutet: Der Schatteneffekt zeigt nach unten rechts.  
 180 bedeutet: Der Schatteneffekt zeigt nach unten.  
 225 bedeutet: Der Schatteneffekt zeigt nach unten links.  
 270 bedeutet: Der Schatteneffekt zeigt nach links.  
 315 bedeutet: Der Schatteneffekt zeigt nach oben links.



### **filter:Wave() (wellenartige Störung)**

Dieser Filter bewirkt eine wellenförmige Verzerrung des betroffenen Text- oder Grafikobjekts. Ein Beispiel:

```
<html><head><title>filter:Wave()/</title>
</head><body bgcolor="#FFFFFF" text="#000000">

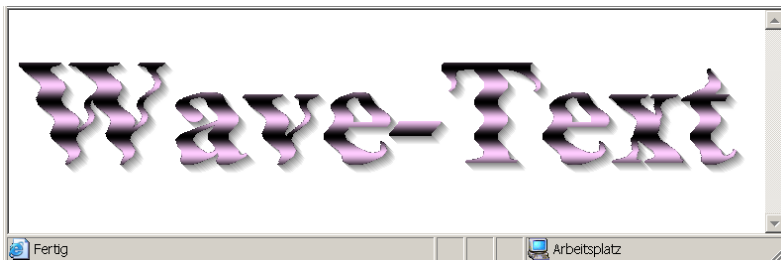
<div style="width:100%; font-size:92pt; font-weight:900; color:#FFCCFF;
```

```
filter:Wave(freq=5, light=20, phase=50, strength=6) Shadow(color=#808080,
direction=135);">
Wave-Text</div>

</body></html>
```

Mit `filter:Wave()` können Sie eine wellenartige Verzerrung definieren. Dieser Filter erwartet folgende Parameter:

- `freq`= Die Wellenfrequenz. Je höher der Wert, desto kleiner die Wellen, je niedriger, desto größer die einzelnen Wellen. Praxisnah sind etwa Werte zwischen 5 und 50.
- `light`= Stärke des Lichts im Welleneffekt in Prozent. Werte zwischen 0 und 100.
- `phase`= Beginn der Sinus-Wellenphase in Prozent. Werte zwischen 0 und 100. Bei 0 beginnt der Welleneffekt normal, bei 25 etwa beginnt er bei 90°.
- `strength`= Stärke des gesamten Effekts. Praxisnah sind z.B. Werte zwischen 1 und 10.
- `add`= Wenn der Effekt auf eine Grafik angewendet wird, bewirkt `add=1`, dass die Originalgrafik zu der verzerrten Grafik hinzugefügt wird. Bei `add=0` wird die Originalgrafik nicht hinzugefügt.



16.63: Die Wirkung des Wave-Filters hängt unter anderem von der Breite des Browser-Fensters ab.



#### **`filter:XRay()` (Foto-Negativ-Effekt)**

Dieser Filter konvertiert alle Farben eines Elements oder einer Grafik in Graustufen und bewirkt einen Effekt wie ein Foto-Negativ. Mit `filter:XRay()` können Sie den Foto-Negativ-Effekt erzwingen. Dieser Filter erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>filter:XRay()</title>
</head><body bgcolor="#FFFFFF" text="#000000">

<p>Links die normale Grafik, rechts mit XRay()-Filter:</p>




</body></html>
```



16.64: Aus Bildern macht der XRay-Filter ein graues Negativ.

# 17 JavaScript und DOM

## 17.1 JavaScript und HTML

JavaScript ist kein direkter Bestandteil von HTML, sondern eine eigene Programmiersprache. Diese Sprache wurde jedoch eigens zu dem Zweck geschaffen, HTML-Autoren ein Werkzeug in die Hand zu geben, mit dessen Hilfe sich Web-Seiten optimieren lassen.

JavaScripts werden wahlweise direkt in der HTML-Datei oder in separaten Dateien notiert. Sie werden zur Laufzeit vom Web-Browser interpretiert. Dazu besitzen moderne Web-Browser entsprechende Interpreter-Software.

Ein paar praktische Beispiele für die Anwendung von JavaScript können Sie in der HTML-Version dieses Buches auf der Buch-CD ausprobieren. So sind beispielsweise die *kleinen Helferlein* JavaScript-Anwendungen, ebenso die *Quickbar* und die *Suche*. Letztere ist allerdings eine ziemlich anspruchsvolle Anwendung, die schon weit über das übliche Maß dessen hinausgeht, wie JavaScript üblicherweise eingesetzt wird.



17.1: Die Suchfunktion auf der Buch-CD liefert schnell die gewünschten Ergebnisse.

In einer Programmiersprache wie JavaScript gibt es für Anfänger viele verwirrende Elemente: Sonderzeichen, Variablen, Wenn-dann-Anweisungen, Schleifen, Funktionen, Methoden, Parameter, Objekte, Eigenschaften und anderes mehr. Um mit diesen Elementen richtig umgehen zu können, müssen Sie lernen sich vorzustellen, was im Computer passiert, wenn der Programmcode ausgeführt wird. Das ist ein langwieriger Lernprozess, den Anfänger nur durch viel Übung bewältigen. JavaScript ist dazu allerdings hervorragend geeignet, weil es eine vergleichsweise



einfache Sprache ist, bei der viele Aufgabenbereiche einer »großen« Programmiersprache fehlen, z.B. Dinge wie Arbeitsspeicherverwaltung oder Dateioperationen. Außerdem setzt JavaScript auf einer bestimmten Umgebung auf, nämlich auf einer anzuzeigenden oder angezeigten Web-Seite.

JavaScript läuft in einer so genannten »Sandbox«. Das ist eine Art Sicherheitskäfig, in dem die Programmiersprache eingesperrt ist. Sie wird dabei um typische Möglichkeiten anderer Programmiersprachen beschnitten, vor allem um die Möglichkeit, beliebig Daten aus Dateien lesen und in Dateien schreiben zu können. So soll verhindert werden, dass JavaScript-Programmierer auf den Rechnern von Anwendern, die eine JavaScript-unterstützte Web-Seite aufrufen, Unfug treiben können. Nach einigen Anfangsproblemen der Vergangenheit funktioniert das mittlerweile auch ganz gut. Wer also JavaScript in seinem Browser deaktiviert aus Angst, dass seine Festplatte damit formatiert werden könnte, sollte sich am besten ganz vom Internet fernhalten, denn schon beim Abruf von E-Mails lauern Gefahren, die wesentlich größer sind als diejenigen, die von JavaScript ausgehen. Doch leider gibt es andere gute Gründe, die nicht wenige Anwender dazu veranlassen, JavaScript in ihrem Browser abzuschalten. Dazu gehören vor allem die »Nervenreien« mancher Programmierhelden, die meinen, eine Web-Seite sei um so toller, je mehr sie den Anwender gängelt. Da wird dann beispielsweise mit JavaScript die Unterstützung der rechten Maustaste abgewürgt, der Anwender kann die aufgerufene Seite nicht mehr verlassen und allerlei mehr. Solche Hirngeburten kontrollgeiler Programmierer sind das eigentliche Übel und der Grund, warum JavaScript nicht bei allen Anwendern beliebt ist. Wer JavaScript einsetzt, sollte sich darüber im Klaren sein und die Möglichkeiten der Sprache so einsetzen, dass der Anwender einen Mehrwert davon hat, und nicht so, dass ihm etwas genommen wird.

Bevor Sie sich also in die Tiefen von JavaScript stürzen, sollten Sie sich auch darüber bewusst sein, dass JavaScript wirklich nur als Ergänzung zu normalem HTML gedacht ist, aber nicht als dessen Ersatz. Es ist zwar durchaus möglich, Web-Seiten ausschließlich in JavaScript zu programmieren, sodass man kaum mehr davon reden kann, dass JavaScript in HTML eingebettet ist, sondern eher davon, dass der HTML-Code der Seite mit Hilfe von JavaScript dynamisch erzeugt wird. Doch solche Seiten funktionieren nur dort, wo JavaScript im Browser aktiviert ist und wo der Browser all das versteht, was Sie programmiert haben (ansonsten endet der Versuch, die Seite anzuzeigen, sehr schnell in einer nüchternen Fehlermeldung). Für Web-Seiten, bei denen Information im Vordergrund steht und die auch von Suchmaschinen-Robots und Benutzerrandgruppen wie Sehbehinderten erfasst werden sollen, müssen Sie darauf achten, JavaScript nur so einzusetzen, dass die Web-Seiten auch ohne eingeschaltetes JavaScript ordentlich angezeigt werden und die Navigation zwischen den Seiten auch ohne JavaScript funktioniert.

## 17.2 Standards, Sprachvarianten und Sprachversionen

**JavaScript** ist eine Sprache, die 1995 von Netscape eingeführt und lizenziert wurde. Es war also eine firmeneigene, proprietäre Angelegenheit, und Netscape diktierte, woraus das »offizielle« JavaScript bestand. Mittlerweile ist die Situation aus verschiedenen Gründen unübersichtlicher geworden – aber JavaScript gibt es nach wie vor und erfreut sich bei Web-Seiten-Gestaltern großer Beliebtheit.

Netscape versuchte schon frühzeitig, JavaScript nicht als willkürliche Eigenentwicklung dastehen zu lassen, sondern der Sprache den Charakter eines Standards zu verleihen. Dazu spannte man die Organisation ECMA (European Computer Manufacturers Association, [www.ecma.ch/](http://www.ecma.ch/)) ein.

Unter dem Namen **ECMAScript** ließ sich Netscape von dieser Organisation JavaScript als Industriestandard deklarieren. In der Spezifikation »ECMA-262« wurde dieser Standard festgeschrieben.

Microsoft konterte auf seine Weise. Der MS Internet Explorer interpretiert zwar JavaScript, daneben aber auch die Microsoft-eigene Sprachvariante **JScript**. Diese Sprache implementierte Microsoft nicht zuletzt, um sich aus den Lizenzvorgaben von Netscape herauszuwinden und das, was bei Netscape unter JavaScript lief, selbstständig erweitern zu können. JScript umfasst die ganze Palette von JavaScript, doch daneben auch noch etliche spezielle Befehle für den Zugriff auf das Dateisystem und Betriebssystem Windows (das Ganze bezeichnet Microsoft als »Windows Scripting Host«). JScript ist insgesamt also viel mächtiger als JavaScript, doch es ist auch für andere Aufgaben gedacht. Es hat eine ganz eigene Versionspolitik, die nichts mit der von JavaScript gemein hat. Bekannter als die JScript-Versionen sind jedoch nach wie vor die JavaScript-Versionen, und wenn Sie JavaScript verwenden, geben Sie in der Regel auch »JavaScript« als Sprache an und nicht »JScript«. Der Internet Explorer kommt damit klar.

- Netscape 2.0 interpretiert den seinerzeit eingeführten JavaScript-Sprachstandard 1.0 (Ende 1995). Der MS Internet Explorer versteht diesen Sprachstandard weitgehend seit der Version 3.0 (1996).
- Netscape interpretiert seit Version 3.0 den JavaScript-Standard 1.1 (ebenfalls 1996). Der MS Internet Explorer interpretiert diesen Sprachumfang von JavaScript weitgehend seit der Produktversion 4.0 (1997).
- Netscape interpretiert seit Version 4.0 den JavaScript-Standard 1.2 (1997). Einige Befehle dieser Sprachversion werden auch vom MS Internet Explorer seit Version 4.0 interpretiert.
- Die JavaScript-Version 1.3 ist in Netscape in den heute am meisten verbreiteten Netscape-Versionen 4.06 bis 4.7 implementiert (1998-2001). Der Internet Explorer versteht diese Sprachversion weitgehend seit Produktversion 5.0.
- In die völlig neu programmierte Netscape-Version 6.0 wurde eine Version 1.5 von JavaScript integriert (2000-2001), die neben einigen wenigen JavaScript-eigenen Neuerungen vor allem das Document Object Model (DOM) umsetzt – mehr dazu weiter unten.

Beim Internet Explorer gibt es zum Teil nicht ganz unbeträchtliche Unterschiede, was die Editionen für die einzelnen Plattformen betrifft. So kann es durchaus sein, dass die Windows-Version eines Internet Explorers mit der Produktversion 5.5 JavaScript-Befehle versteht, die eine Macintosh-Ausgabe mit gleicher Produktversionsnummer nicht versteht. Dazu kommen andere Browser-Produkte, die ebenfalls JavaScript implementiert haben, wie beispielsweise Opera, wobei auch dort jede Produktversion wieder ein bisschen mehr JavaScript kann als die Vorgängerversion. In der Praxis ist eigentlich nicht mehr wirklich nachvollziehbar, was nun welcher Browser genau kann. Es ist deshalb ratsam, benutzte Sprachbestandteile von JavaScript-Versionen jenseits der V1.2 besonders sorgfältig zu testen.

Das Gerangel zwischen Netscape und Microsoft um die bessere JavaScript-Unterstützung bewirkte auf jeden Fall eine Menge Frustration bei Web-Programmierern und Anwendern. Viele Dinge mussten und müssen immer noch ziemlich umständlich programmiert werden, weil die Fähigkeiten und Eigenheiten der verschiedenen JavaScript-Interpreter der verschiedenen Browser und Browser-Versionen lauter Wenss und Abers erfordern, was sich im Quellcode von

JavaScript niederschlägt. Vor allem bei komplexeren JavaScript-Anwendungen, etwa bei dynamischem HTML, wurde das »Doppeltprogrammieren« zur lästigen Pflicht.

Um diesem Missstand auf Dauer abzuhelpfen, wurde das W3-Konsortium eingeschaltet, damit für eine allgemeine Sprachnorm gesorgt wird. Das W3-Konsortium erarbeitete dabei jedoch keinen konkreten JavaScript-Standard, sondern ein allgemeines Modell für Objekte eines Dokuments. Dieses Modell sollte eine Script-Sprache, die sich als Ergänzungssprache zu Auszeichnungssprachen wie HTML versteht, vollständig umsetzen. Vom W3-Konsortium erhielt das Modell die Bezeichnung **Document Object Model (DOM)**. Am 1. Oktober 1998 wurde das DOM eine offizielle W3-Empfehlung (*recommendation*) in der Version 1.0, die seit dem 29. September 2000 in einer zweiten Ausgabe vorliegt. Seit 13. November 2000 ist die Version 2.0 des DOM eine offizielle W3-Empfehlung. Das entsprechende Dokument ist jedoch aufgesplittet in mehrere Einzeldokumente. Version 3.0 ist in Arbeit.

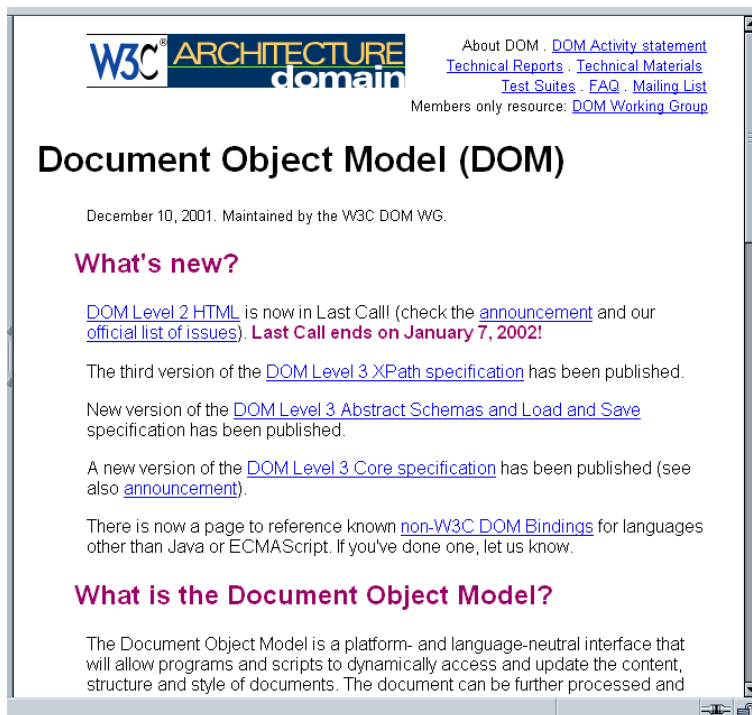
Der MS Internet 5.0 unterstützt einiges von DOM 1.0, die Version 5.5 schon mehr, ebenso wie Netscape 6.x. Die JavaScript-Version, die DOM erstmals umsetzt, ist die Version 1.5.

Ein wichtiger Aspekt des DOM ist, dass es nicht nur für HTML-ergänzende Script-Sprachen konzipiert ist, sondern auch für Script-Sprachen, die jegliche XML-gerechte Auszeichnungssprache erweitern sollen. Das DOM orientiert sich daher in seinem Kern nicht mehr an bestimmten HTML-Elementen. Stattdessen geht es in abstrakter Weise von einem »Dokument« aus (das in XHTML geschrieben sein kann, aber auch in einer anderen XML-basierten Sprache). Es definiert, wie sich auf die einzelnen Elemente dieses Dokuments zugreifen lässt und welche Manipulationsmöglichkeiten es gibt. Da man aber bei der Entwicklung des DOM auch auf die Gegebenheiten und die Verbreitung von HTML und JavaScript Rücksicht nehmen wollte, wurde dem so genannten »Kern-DOM« ein »HTML-Anwendungs-DOM« zur Seite gestellt. Letzteres versucht, Teile des »gewachsenen« JavaScript und MS-JScript zu sanktionieren und Bestandteile des »Kern-DOM« darin zu integrieren.

DOM 1.0 grenzt jedoch noch einige wichtige Aspekte aus, wie etwa das Event-Handling (Behandlung von Anwenderereignissen wie Mausklicks, Tastatureingaben usw.) oder die Integration von Stylesheets in das, was Programmierer dynamisch verändern können. In den Versionen 2.0 und 3.0 behandelt das DOM auch diese Aspekte.

Seit das DOM vorliegt, überbieten sich die Browser-Anbieter gegenseitig mit Versprechungen, die Vorgaben des DOM in JavaScript umzusetzen. Abgesehen davon, dass es noch immer Uneinheitlichkeiten bei der Umsetzung gibt, ähnlich wie in HTML und CSS, haben JavaScript-Programmierer nun ein zusätzliches Problem: Wo sie zunächst häufig eine Doppellösung für die beiden verbreitetsten Browser-Typen benötigten, müssen sie nun manchmal sogar drei Lösungen programmieren, wenn sie nämlich Rücksicht auf die 4er-Versionen von Netscape und Internet Explorer Rücksicht nehmen und gleichzeitig DOM-gerecht programmieren wollen.

In der Praxis von JavaScript auf HTML-basierten Web-Seiten ist es jedoch so, dass meistens noch mit den herkömmlichen, von Netscape eingeführten JavaScript-Objekten gearbeitet wird. In Zukunft ist allerdings zu hoffen, dass die DOM-gerechte JavaScript-Programmierung das klassische JavaScript verdrängen wird.



17.2: Über die W3-Einstiegsseite zum DOM ([www.w3.org/DOM/](http://www.w3.org/DOM/)) können Sie die aktuelle Weiterentwicklung des Document Object Models beim W3-Konsortium verfolgen.

Zur Kennzeichnung der Sprachversionen von JavaScript und von DOM werden in der vorliegenden Dokumentation folgende Symbole verwendet:

| Symbol                                                                              | Bedeutung                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Bestandteil der ersten JavaScript-Version                                                                                                                                                                                                                                                                                                              |
|  | Bestandteil der JavaScript-Version 1.1                                                                                                                                                                                                                                                                                                                 |
|  | Bestandteil der JavaScript-Version 1.2                                                                                                                                                                                                                                                                                                                 |
|  | Bestandteil der JavaScript-Version 1.3                                                                                                                                                                                                                                                                                                                 |
|  | Bestandteil der JavaScript-Version 1.5                                                                                                                                                                                                                                                                                                                 |
|  | Von Microsoft eingeführter Sprachbestandteil. Solche Bestandteile werden in der Regel nur vom MS Internet Explorer ab V4.x interpretiert; es wurden nur einige wichtige solcher Elemente in die Objekt-Referenz mit aufgenommen, nämlich solche, die eine Grundlage für Dynamisches HTML nach dem Ansatz des MS Internet Explorers ab V4.0 darstellen. |
|  | Objekt, Eigenschaft oder Methode nach den Vorgaben des Document Object Model, Version 1.0. Kommt innerhalb der vorliegenden Dokumentation in der Objekt-Referenz vor, um Objekte, Eigenschaften und Methoden auszuzeichnen.                                                                                                                            |

| Symbol                                                                            | Bedeutung                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Objekt, Eigenschaft oder Methode nach den Vorgaben des Document Object Model, Version 2.0. Kommt innerhalb der vorliegenden Dokumentation in der Objekt-Referenz vor, um Objekte, Eigenschaften und Methoden auszuzeichnen. |

### 17.3 Hinweise für die Praxis

Bevor Sie daran gehen, neuen, eigenen JavaScript-Code zu programmieren, müssen Sie sich exakt darüber im Klaren sein, welches Problem Sie damit lösen wollen. Dazu müssen Sie erst einmal wissen, was man mit HTML selbst alles machen kann und wo die Grenzen von HTML liegen. Von JavaScript müssen Sie dann so viel wissen, dass Sie entscheiden können, ob das Problem mit JavaScript überhaupt lösbar ist. Mit JavaScript können Sie z.B. **nicht** die typischen Aufgaben von CGI lösen! JavaScripts werden im Browser des Anwenders ausgeführt, nicht auf dem Server, wo die Web-Seiten abgelegt sind. Das heißt, JavaScript kann vom Browser erst dann ausgeführt werden, wenn er eine Web-Seite gerade einliest oder nachdem er sie eingelesen hat – dann z.B. auch ausgelöst durch Benutzerereignisse wie Mausklicks usw.

Aus diesem Grund können Sie mit JavaScript also keine Anwendungen wie Gästebücher oder Web-Foren programmieren. Denn solche Anwendungen müssen die Daten aller beitragenden Anwender zentral auf dem Server speichern. Wohl aber kann JavaScript unterstützend eingesetzt werden bei solchen Anwendungen. So ist es beispielsweise sinnvoller, Formulareingaben des Anwenders vor dem Absenden mit Hilfe von JavaScript zu überprüfen, als den Server-Rechner mit dieser Arbeit zusätzlich zu belasten.

Wenn Sie etwas mit JavaScript lösen möchten, lohnt es sich durchaus, sich im Web umsehen, ob es nicht schon frei verfügbare JavaScript-Beispiele gibt, die genau Ihr Problem lösen. Denn es ist immer besser, auf Code zurückzugreifen, der sich im Einsatz bereits bewährt hat, als neuen Code zu erstellen, dessen »heimliche Tücken« einem noch nicht bekannt sind. In vielen Fällen genügt es, vorhandene JavaScripts den eigenen Erfordernissen anzupassen. Bei vorhandenen JavaScripts müssen Sie allerdings so viel von der Sprache verstehen, dass Sie genau wissen, welche Variablen oder festen Werte Sie verändern oder welche Funktion Sie ergänzen wollen.

Im Linkverzeichnis des Online-Angebots von SELFHTML ([selfaktuell.teamone.de](http://selfaktuell.teamone.de)) aktuell finden Sie Links zu Anbietern frei verfügbarer JavaScripts. Wenn Sie JavaScript auf eigenen Web-Seiten verwenden wollen, sollten Sie auf jeden Fall mehrere Browser zum Testen einsetzen. Denn leider sind die JavaScript-Interpreter der JavaScript-fähigen WWW-Browser wie schon erwähnt sehr unterschiedlich in ihrer Leistung.

Stellen Sie keine Web-Seiten mit ungeprüfem JavaScript-Code öffentlich zur Verfügung. Für einen Anwender ist es sehr ärgerlich, wenn er Fehlermeldungen des JavaScript-Interpreters am Bildschirm erhält und in schlimmeren Fällen einen Programmabsturz des Web-Browsers oder gar einen Systemabsturz hat, was bei Online-Sitzungen besonders unangenehm ist. Solche Anwender besuchen Ihre Seiten mit Sicherheit so schnell nicht wieder.

## 17.4 JavaScript-Bereiche in HTML definieren

Für JavaScript-Programmabschnitte können Sie in HTML Bereiche definieren. Ein Beispiel:

```
<html>
<head><title>Test</title>
<script type="text/javascript">
<!--
  alert("Hallo Welt!");
//-->
</script>
</head>
<body>
</body>
</html>
```



17.3: Die minimale JavaScript-Anwendung aus dem Beispiel öffnet eine Dialogbox.

Mit `<script type="text/javascript">` leiten Sie einen Bereich für JavaScript innerhalb einer HTML-Datei ein (*script* = *Quelltext*, *type* = *Mime-Type*). Die Angabe zum Mime-Type ist seit HTML 4.0 Pflicht. Mit `type="text/javascript"` bestimmen Sie den typischen Mime-Type für JavaScript-Dateien.

Zusätzlich können Sie zur Kennzeichnung der JavaScript-Sprachversion das `language`-Attribut verwenden (z.B. `language="JavaScript"` oder `language="JavaScript1.2"`). Jedoch gehört dieses Attribut zu den missbilligten Attributen und wird auch nicht von allen Browsern korrekt interpretiert.

Dahinter – am besten in der nächsten Zeile – sollten Sie mit `<!--` einen HTML-Kommentar einleiten. Dadurch erreichen Sie, dass ältere WWW-Browser, die JavaScript nicht kennen, den folgenden JavaScript-Code ignorieren und nicht irrtümlich als Text innerhalb der HTML-Datei interpretieren.

Den JavaScript-Bereich schließen Sie mit einem einzeiligen JavaScript-Kommentar `//`, gefolgt vom schließenden HTML-Kommentar `-->` und `</script>` ab. Der JavaScript-Kommentar ist erforderlich, um Fehlermeldungen in scriptfähigen Browsern zu unterdrücken. Berücksichtigen Sie, dass vor und nach dem HTML-Kommentar ein Zeilenumbruch zwingend erforderlich ist.

**Beachten Sie:** Es gibt keine festen Vorschriften dafür, an welcher Stelle einer HTML-Datei ein JavaScript-Bereich definiert werden muss. Es ist unter JavaScript-Programmierern zur Gewohnheit geworden, einen solchen Bereich im Kopf der HTML-Datei, also zwischen `<head>` und `</head>` zu definieren. Dadurch ist am ehesten sichergestellt, dass der Code vom WWW-Browser bereits eingelesen ist und zur Verfügung steht, wenn er ausgeführt werden soll.

JavaScript-Code kann automatisch beim Einlesen der HTML-Datei ausgeführt werden. Das ist dann der Fall, wenn JavaScript-Befehle in einem JavaScript-Bereich außerhalb jeder selbst definierten Funktion stehen, so wie im obigen Beispiel. In solchen Fällen ist es manchmal auch erforderlich, den Script-Bereich innerhalb des HTML-Dateikörpers, also innerhalb von `<body>...</body>`, zu notieren, zum Beispiel wenn Sie am Ende der Datei mit JavaScript dynamisch Datum und Uhrzeit der letzten Änderung an der Datei schreiben wollen.

## 17.5 JavaScript-Anweisungen in HTML-Tags

JavaScript kann auch innerhalb herkömmlicher HTML-Tags vorkommen. Das ist dann kein komplexer Programmcode, sondern in der Regel nur der Aufruf bestimmter Methoden, Funktionen, Objekte, Eigenschaften. Für den Aufruf gibt es so genannte Event-Handler (Kapitel 18.10). Das sind Attribute in HTML-Tags, über die sich JavaScripts aktivieren lassen. Für jeden der möglichen Event-Handler ist festgelegt, in welchen HTML-Tags er vorkommen darf. Das Ergebnis des folgenden Beispiel sehen Sie in Kapitel 17.6 als Bild:

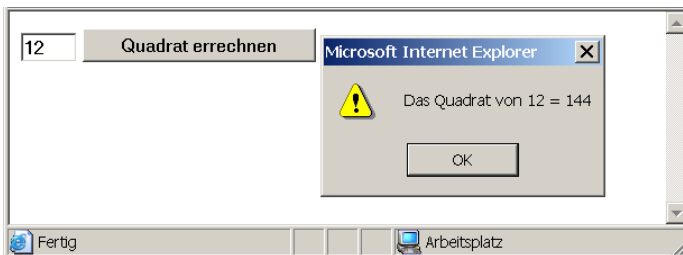
```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function Quadrat() {
    var Ergebnis = document.Formular.Eingabe.value *
document.Formular.Eingabe.value; alert("Das Quadrat von " +
document.Formular.Eingabe.value + " = " + Ergebnis);
}
//-->
</script>
</head>
<body>
<form name="Formular" action="">
<input type="text" name="Eingabe" size="3">
<input type="button" value="Quadrat errechnen" onClick="Quadrat()">
</form>
</body>
</html>
```

Das obige Beispiel zeigt eine komplette HTML-Datei. Im Dateikopf ist ein JavaScript-Bereich definiert. Innerhalb dieses Bereichs steht eine selbst definierte Funktion, die Funktion mit dem Namen `Quadrat()`. Die Funktion errechnet das Quadrat einer Zahl, die der Anwender in dem weiter unten notierten Formular im Eingabefeld mit dem Namen `Eingabe` eingibt. Der JavaScript-Code dieser Funktion wird aber nicht automatisch ausgeführt, sondern nur dann, wenn die Funktion explizit aufgerufen wird. Im obigen Beispiel erfolgt der Aufruf mit Hilfe eines Klick-Buttons. Wenn der Anwender auf den Button klickt, wird das Quadrat zur eingegebenen Zahl ausgegeben. Dazu ist im HTML-Tag des Klick-Buttons das Attribut `onClick=` notiert – ein Event-Handler mit der Bedeutung »beim Anklicken«.

**Beachten Sie:** Im Abschnitt 18.19 Event-Handler wird beschrieben, welche Event-Handler es außer `onClick=` noch gibt und in welchen HTML-Tags sie vorkommen dürfen.

## 17.6 JavaScript in separaten Dateien

Seit der Sprachversion 1.1 von JavaScript (und auch seit HTML 4.0) ist es möglich, JavaScript-Code in separaten Dateien zu notieren. Das ist sehr nützlich, wenn Sie gleiche JavaScript-Funktionen in mehreren HTML-Dateien verwenden wollen. Denn so brauchen Sie den Code nur einmal notieren und können ihn in mehreren HTML-Dateien referenzieren.



17.4: Das Beispiel berechnet das Quadrat einer Zahl und zeigt das Ergebnis als Dialogbox an.

Beispiel: JavaScript-Datei *quadrat.js*

```
function Quadrat() {
    var Ergebnis = document.Formular.Eingabe.value *
document.Formular.Eingabe.value; alert("Das Quadrat von " +
document.Formular.Eingabe.value + " = " + Ergebnis);
}
```

Beispiel: HTML-Datei

```
<html>
<head>
<title>JavaScript-Test</title>
<script src="quadrat.js" type="text/javascript">
</script>
</head>
<body>
<form name="Formular" action="">
<input type="text" name="Eingabe" size="3">
<input type="button" value="Quadrat errechnen" onClick="Quadrat()">
</form>
</body>
</html>
```

Hier handelt es sich um das gleiche Beispiel wie im Abschnitt zuvor. Der Unterschied ist lediglich, dass der JavaScript-Code in einer separaten Datei steht. Dazu notieren Sie im einleitenden `<script>`-Tag die Angabe `src=` (*src = source = Quelle*). Als Wert weisen Sie den URI der sepa-



raten Datei mit dem Quellcode zu. Dabei gelten die Regeln zum Referenzieren in HTML. Die Datei mit dem Quellcode muss eine reine ASCII-Datei sein und sollte die Dateinamenerweiterung `.js` erhalten. Die Datei darf nichts anderes als JavaScript-Code enthalten.

**Beachten Sie:** Es kann passieren, dass ein JavaScript, das in einer separaten Datei notiert ist, lokal wunderbar funktioniert, aber nach dem Hochladen der Dateien auf einen WWW-Server plötzlich nicht mehr. Bitten Sie in diesem Fall den Provider oder Webmaster des Servers, den Mime-Type `text/javascript` für Dateien mit der Endung `.js` in die Konfiguration des Web-Servers mit aufzunehmen. Ansonsten können Sie im einleitenden `<script>`-Tag beim `type`-Attribut auch mal den Mime-Type `application/x-javascript` anstelle von `text/javascript` ausprobieren. Bei einigen Servern funktioniert es dann.

# 18 JavaScript: Sprachelemente

## 18.1 Allgemeine Regeln für JavaScript

### 18.1.1 Anweisungen notieren

JavaScript besteht letztendlich aus einer kontrollierten Anordnung von Anweisungen. Das sind Befehle, die der JavaScript-Interpreter des WWW-Browsers bewertet und in Maschinencode umsetzt, der auf dem Rechner des Anwenders ausführbar ist. Es gibt einfache und komplexere Anweisungen.

Beispiel 1:

```
Zahl = 42;
```

Beispiel 2:

```
Quadrat = Zahl * Zahl;
```

Beispiel 3:

```
if(Zahl > 1000)
  Zahl = 0;
```

Beispiel 4:

```
alert("Das Quadrat von " + Zahl + " = " + Ergebnis);
```

Eine Anweisung in JavaScript besteht immer aus einem Befehl, der mit einem Strichpunkt ; oder einem Zeilenumbruch abgeschlossen wird. In neueren Netscape-Dokumentationen zu JavaScript wird der Strichpunkt am Ende von einfachen Anweisungen zwar häufig weggelassen, aber um unnötige Fehler zu vermeiden, ist es ratsam, sich von vornherein anzugewöhnen, alle Anweisungen auf diese Weise abzuschließen. Eine Anweisung ist zum Beispiel,

- wenn Sie einer Variablen einen Wert zuweisen, wie in Beispiel 1;
- wenn Sie mit Variablen oder Werten eine Operation durchführen, wie in Beispiel 2;
- wenn Sie einen Befehl nur unter bestimmten Bedingungen ausführen, wie in Beispiel 3 (siehe hierzu die Abschnitte über bedingte Anweisungen und über Schleifen, Kapitel 18.7 und 18.8);
- wenn Sie eine selbst definierte Funktion oder eine Objektmethode aufrufen bzw. wenn Sie eine Objekteigenschaft auslesen oder ändern (siehe hierzu den Abschnitt 18.3 *Objekte, Eigenschaften und Methoden*), wie in Beispiel 4.

### 18.1.2 Anweisungsblöcke notieren

Ein Anweisungsblock besteht aus einer oder mehreren Anweisungen, die innerhalb einer übergeordneten Anweisung oder innerhalb einer Funktion stehen. So können Anweisungsblöcke beispielsweise unterhalb einer bedingten Anweisung oder einer Schleife stehen. Auch alle Anweisungen, die innerhalb einer selbst definierten Funktion stehen, bilden einen Anweisungsblock.

Beispiel 1:

```
if(Zahl > 1000) {
    Zahl = 0;
    Neustart();
}
```

Beispiel 2:

```
while(i <= 99) {
    Quadrat(i);
    i = i + 1;
}
```

Beispiel 3:

```
function SageQuadrat(x) {
    var Ergebnis = x * x;
    alert(Ergebnis);
}
```

Beispiel 4:

```
function SagEinmaleins(x) {
    var Ergebnis = x * x;
    if(Ergebnis > 100) {
        Ergebnis = 0;
        Neustart();
    }
    alert(Ergebnis);
}
```

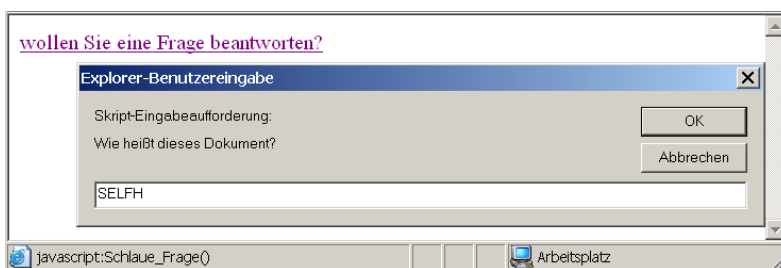
Ein Anweisungsblock wird durch eine öffnende **geschweifte** Klammer { begonnen und durch eine schließende geschweifte Klammer } beendet. Sie können die geschweiften Klammern jeweils in eine eigene Zeile schreiben, so wie in den obigen Beispielen. Es ist aber auch erlaubt, die Klammern in der gleichen Zeile zu notieren wie die Anweisungen.

Bei bedingten Anweisungen (wie in Beispiel 1) oder bei Schleifen (wie in Beispiel 2) müssen Sie solche Anweisungsblöcke notieren, sobald mehr als eine Anweisung von der Bedingung oder der Schleifenbedingung abhängig ausgeführt werden soll. Bei Funktionen (wie in Beispiel 3) müssen Sie Anfang und Ende der Funktion durch geschweifte Klammern markieren. Alles, was innerhalb der Funktion steht, ist daher ein Anweisungsblock. Anweisungsblöcke können auch verschachtelt sein, so wie in Beispiel 4 oben.

### 18.1.3 Selbst vergebene Namen

An vielen Stellen in JavaScript müssen Sie selbst Namen vergeben, zum Beispiel für selbst definierte Funktionen, eigene Objekte oder Variablen. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function Schlaue_Frage() {
  var i = 1;
  var Eingabe = "";
  while(Eingabe != "SELFHTML" && i <= 3) {
    Eingabe = window.prompt("Wie heißt dieses Dokument?", "");
    i++;
  }
}
//-->
</script>
</head>
<body>
<a href="javascript:Schlaue_Frage()">wollen Sie eine Frage beantworten?</a>
</body>
</html>
```



18.1: Die Dialogbox erfragt eine Zeichenkette vom Benutzer.

Im Beispiel ist in einem JavaScript-Bereich eine Funktion mit Namen `Schlaue_Frage` notiert. Darin werden zwei Variablen mit den Namen `i` und `Eingabe` deklariert und verwendet. Das Beispiel ruft beim Anklicken des HTML-Verweises die Funktion `Schlaue_Frage` auf. Darin wird ein Dialogfenster ausgegeben, in dem der Anwender bis zu dreimal auf die Frage antworten kann, wie dieses Dokument heißt. Namen vergeben müssen Sie also recht häufig in JavaScript.

Bei selbst vergebenen Namen gelten folgende Regeln:

- Sie dürfen keine Leerzeichen enthalten.

- Sie dürfen nur aus Buchstaben und Ziffern bestehen – das erste Zeichen muss ein Buchstabe sein; es sind Groß- und Kleinbuchstaben erlaubt. Groß- und Kleinschreibung werden unterschieden!
- Sie dürfen keine deutschen Umlaute oder scharfes S enthalten.
- Sie dürfen als einziges Sonderzeichen den Unterstrich »\_« enthalten.
- Sie dürfen nicht mit einem reservierten Wort identisch sein (Kapitel 18.9).

Vergeben Sie sprechende Namen, die Ihnen auch ein halbes Jahr, nachdem Sie das Script geschrieben haben, noch signalisieren, welche Bedeutung sie haben. Es dürfen ruhig auch deutschsprachige Wörter sein, solange die genannten Regeln eingehalten werden.

### 18.1.4 Kommentare in JavaScript

Bei komplexeren Programmteilen können Sie Kommentare benutzen, um einzelne Anweisungen zu erklären. Auch wenn Sie Ihre Copyright-Angaben innerhalb eines selbst geschriebenen JavaScript-Codes unterbringen wollen, können Sie dies mit Hilfe eines Kommentars tun. Kommentare werden vom JavaScript-Interpreter des WWW-Browsers ignoriert. Ein Beispiel:

```
while(i <= 99) {
    Quadrat = i * i;      /* solange i kleiner gleich 99, Quadrat von i bilden */
    i = i + 1;           // i um eins erhoeihen
}
```

Einen wahlweise ein- oder mehrzeiligen Kommentar leiten Sie mit `/*`, also einem Schrägstrich, gefolgt von einem Stern-Zeichen, ein. Mit der umgekehrten Folge `*/`, also einem Stern-Zeichen, gefolgt von einem Schrägstrich, beenden Sie den Kommentar. Einen einzeiligen Kommentar starten Sie mit der Zeichenfolge `//`. Der Browser ignoriert dann den nachfolgenden Text bis zum nächsten Zeilenende.

## 18.2 Variablen und Werte

### 18.2.1 Variablen definieren

Variablen sind Speicherbereiche, in denen Sie Daten, die Sie im Laufe Ihrer Programmprozeduren benötigen, speichern können. Der Inhalt, der in einer Variablen gespeichert ist, wird als »Wert« bezeichnet. Sie können den Wert einer Variablen jederzeit ändern. Um mit Variablen arbeiten zu können, müssen Sie die benötigten Variablen zuerst definieren. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
    var Hinweis = "Gleich werden Quadratzahlen ausgegeben";
    alert(Hinweis);
```

```

function SchreibeQuadrate() {
    var SinnDesLebens = 42;
    var i, x;
    var Satzteil = "Das Quadrat von ";
    for(i=1; i <= SinnDesLebens; ++i) {
        x = i * i;
        document.write(Satzteil + i + " ist " + x + "<br>");
    }
}
//-->
</script>
</head>
<body onLoad="SchreibeQuadrate()">
</body>
</html>

```



18.2: Das Beispielscript gibt eine Liste mit Quadratzahlen aus.

Das Beispiel gibt beim Aufruf der Seite eine Meldung aus, dass gleich Quadratzahlen ausgegeben werden. Bestätigt der Anwender das Meldungsfenster, werden die Quadrate der Zahlen von 1 bis 42 ausgegeben.

Es gibt **globale Variablen** und **lokale Variablen**. Eine lokale Variable erhalten Sie durch die Deklaration der Variablen mit `var` innerhalb einer Funktion. Im obigen Beispiel sind die Variablen `SinnDesLebens`, `i`, `x` und `Satzteil` innerhalb der Funktion `SchreibeQuadrate()` als lokale Variablen notiert. Diese Variablen sind deshalb nur innerhalb dieser Funktion gültig. Man spricht in diesem Zusammenhang auch von der »Lebensdauer« von Variablen. Parameter, die einer Funktion übergeben werden, werden ebenfalls als lokale Variablen behandelt. Die Variable `Hinweis` ist dagegen eine globale Variable. Sie ist im gesamten Dokument gültig und steht jederzeit zur Verfügung. Wenn Sie innerhalb von Funktionen Variablen **ohne** das Schlüsselwort `var` deklarieren, dann sind diese Variablen global.

Jede Variablendeklaration wird mit einem Strichpunkt abgeschlossen, da sie eine ganz normale Anweisung darstellt.

Variablen können mit oder ohne weitere Wertzuweisung deklariert werden. Im obigen Beispiel wird etwa der Variablen `SinnDesLebens` bei der Definition gleich ein Wert zugewiesen, nämlich 42. Auch die Variable `Satzteil` erhält eine anfängliche Wertzuweisung, nämlich den Wert "Das Quadrat von ". Die Variablen `i` und `x` werden dagegen nicht mit einem Anfangswert versehen. Beim Zuweisen eines Wertes notieren Sie hinter dem Variablennamen ein Gleichheitszeichen und dahinter den Wert, den Sie der Variablen zuweisen wollen.

Sie können mehrere Variablen auf einmal definieren, so wie die beiden Variablen `i` und `x` im Beispiel. Dazu trennen Sie die Variablennamen durch Kommata. Das ist natürlich auch in Verbindung mit zugewiesenen Anfangswerten möglich.

Es gibt numerische Variablen und Variablen für Zeichen bzw. Zeichenketten. Im obigen Beispiel sind die Variablen `SinnDesLebens`, `i` und `x` numerische Variablen. Die Variablen `Hinweis` und `Satzteil` sind dagegen Zeichenkettenvariablen. Dies ist daran erkennbar, dass der ihnen zugewiesene Wert, ein Text, in Anführungszeichen gesetzt wird. Sie könnten z.B. eine Variable `Nummer = 1`; und eine Variable `Zeichen = "1"`; definieren. Der Unterschied ist, dass Sie mit der Variablen `Nummer` Rechenoperationen anstellen können, mit der Variablen `Zeichen` nicht. Dafür können Sie mit Zeichenkettenvariablen Zeichenkettenoperationen durchführen, etwa das Extrahieren einer Teilzeichenkette, was mit numerischen Variablen nicht möglich ist.

**Beachten Sie:** Bei der Vergabe von Variablennamen gelten die Regeln für selbst vergebene Namen (Kapitel 21.1.3). Variablen in JavaScript sind nicht so streng »getypt« wie in vielen anderen Programmiersprachen. Einfache Variablentypen, wie Zahlen, Zeichenketten oder Wahrheitswerte, werden lediglich nach numerischen und nichtnumerischen Variablen eingeteilt. Kommazahlen und Ganzzahlen benötigen keine unterschiedlichen Typen. Der Inhalt von numerischen Variablen kann ohne vorherige Konvertierung in Zeichenketten auf den Bildschirm oder in Meldungsfenster geschrieben werden. Umgekehrt können Sie aber mit Zeichenketten, z.B. Werten aus Formularfeldern, nicht immer automatisch rechnen, sondern müssen sie vorher explizit in Zahlen umwandeln. Für die explizite Typumwandlung gibt es verschiedene objektunabhängige Funktionen (Kapitel 20.33).

## 18.2.2 Werte von Variablen ändern

Wertänderungen von Variablen sind das A & O bei der Programmierung. Sie werden nur dann erfolgreich eigene Programme schreiben können, wenn Sie jederzeit den Überblick haben, was in einer Variablen an einem bestimmten Punkt des Programmablaufs steht. Besonders, wenn Sie mit bedingten Anweisungen oder Schleifen arbeiten, werden Sie schnell feststellen, wie leicht der Überblick über den aktuellen Zustand einer Variablen verloren gehen kann. Ein Beispiel:

```
<html>
<head>
<title>Sinn des Lebens zum Quadrat</title>
<script type="text/javascript">
<!--
    function SchreibeTabelle() {
        var SinnDesLebens = 42;
        var i, x, y;
        var Satzteil = "Das Quadrat von ";
        document.close();
```

```

document.open("text/html");
document.writeln('<table border="1"><tr>');
document.writeln('<td bgcolor="EEEEEE">Wert</td>');
document.writeln('<td bgcolor="EEEEEE">Wert<sup>2</sup></td>');
document.writeln('<td bgcolor="EEEEEE">Wert<sup>3</sup></td></tr>');
for(i=1; i <= SinnDesLebens; ++i) {
    x = i * i;
    y = i * i * i;
    document.writeln('<tr><td>' + i + '</td>');
    document.writeln('<td>' + x + '</td>');
    document.writeln('<td>' + y + '</td></tr>');
}
document.writeln('</table>');
}
//-->
</script>
</head>
<body onLoad="SchreibeTabelle()">
</body>
</html>

```

Das Beispiel leert das Anzeigefenster des Browsers und erzeugt dynamisch eine Tabelle mit 3 Spalten und 42 Zeilen plus einer Kopfzeile. In die Zellen der Tabelle werden für die Zahlen von 1 bis 42 der Quadratwert und der Kubikwert geschrieben. Zum dynamischen Schreiben in das Browser-Fenster benutzt das Script Methoden des Objekts `document`.

Die Variablen `SinnDesLebens` und `Satzteil` werden während des Programmablaufs zwar benutzt, aber ihr Wert wird nicht geändert. Die Variablen `i`, `x` und `y` dagegen ändern ihren Wert laufend. Das liegt daran, dass sie innerhalb einer `for`-Schleife bei jedem Schleifendurchlauf neue Werte zugewiesen bekommen.

Die Wertzuweisung erfolgt, indem Sie den Variablennamen, dahinter ein Gleichheitszeichen und dahinter den gewünschten Wert notieren. Bei dem Wert, den Sie zuweisen, können Sie anstelle einer bestimmten Zahl oder einer Zeichenkette auch Namen anderer Variablen notieren. So wird im Beispiel der Variablen `x` bei jedem Schleifendurchlauf als Wert das Ergebnis der mit sich selbst multiplizierten Variablen `i` zugewiesen und `y` das Ergebnis von `i * i * i`.

## 18.3 Objekte, Eigenschaften und Methoden

### 18.3.1 Vordefinierte JavaScript-Objekte

Objekte sind fest umgrenzte Datenelemente mit Eigenschaften und oft auch mit objektgebundenen Funktionen (Methoden). Dass JavaScript eine Erweiterung von HTML darstellt, liegt vor allem an den vordefinierten Objekten, die Ihnen in JavaScript zur Verfügung stehen. Über diese vordefinierten Objekte können Sie beispielsweise einzelne Elemente eines HTML-Formulars abfragen oder ändern.



Ein Objekt kann eine Teilmenge eines übergeordneten Objekts sein. Die JavaScript-Objekte sind deshalb in einer Hierarchie geordnet. Das hierarchiehöchste Objekt ist in JavaScript das Fenster-Objekt (`window`). Fenster haben Eigenschaften wie einen Namen, eine Größe usw. Der Inhalt eines Fensters ist das nächstniedrigere Objekt, nämlich das im Fenster angezeigte Dokument (in JavaScript das Objekt `document`). In der Regel ist der Fensterinhalt eine HTML-Datei. Eine solche Datei kann bestimmte, durch HTML-Tags definierte Elemente enthalten, wie zum Beispiel Formulare, Verweise, Grafikreferenzen usw. Für solche untergeordneten Elemente gibt es wieder eigene JavaScript-Objekte, zum Beispiel das Objekt `forms` für Formulare. Ein Formular besteht seinerseits aus verschiedenen Elementen, zum Beispiel aus Eingabefeldern, Auswahllisten oder Buttons zum Absenden bzw. Abbrechen. In JavaScript gibt es dafür ein Objekt `elements`, mit dem Sie einzelne Felder und andere Elemente innerhalb eines Formulars ansprechen können.

Neben den hierarchisch geordneten JavaScript-Objekten gibt es auch solche, die nicht direkt in die Hierarchie passen. Das sind zum Beispiel Objekte für Datums- und Zeitrechnung, für mathematische Aufgaben oder für Zeichenkettenverarbeitung. Eine Übersicht der vordefinierten JavaScript-Objekte finden Sie in Kapitel 20.

### 18.3.2 Vordefinierte JavaScript-Objekte verwenden

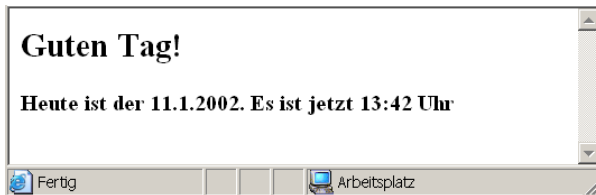
JavaScript-Objekte können Sie jederzeit verwenden. Ein Beispiel:

```
<html><head><title>Test</title>
</head>
<body>
<script type="text/javascript">
<!--
    var Jetzt = new Date();
    var Tag = Jetzt.getDate();
    var Monat = Jetzt.getMonth() + 1;
    var Jahr = Jetzt.getYear();
    var Stunden = Jetzt.getHours();
    var Minuten = Jetzt.getMinutes();
    var NachVoll = ((Minuten < 10) ? ":0" : ":");
    if (Jahr<2000) Jahr=Jahr+1900;
    document.write("<h2>Guten Tag!\</h2><b>Heute ist der "
+ Tag + "." + Monat + "." + Jahr + ". Es ist jetzt "
+ Stunden + NachVoll + Minuten + " Uhr</b>");
-->
</script>
</body></html>
```

Im Beispiel wird innerhalb eines JavaScript-Bereichs mit Hilfe des vordefinierten JavaScript-Objekts `Date` das aktuelle Datum und die aktuelle Uhrzeit ermittelt und HTML-formatiert ausgegeben.

Zuerst muss dazu eine neue Variable angelegt werden. Im Beispiel ist das die Variable `Jetzt`. Diese Variable soll auf Daten des `Date`-Objekts zugreifen können. Dies geschieht durch ein Gleichheitszeichen hinter dem Variablennamen. Dahinter folgt das reservierte JavaScript Wort

`new`, und dahinter, durch ein Leerzeichen getrennt, der Aufruf der von `Date()` zum Erzeugen einer neuen Instanz des Objekts `Date`.



18.3: Datum und Uhrzeit entnimmt der Browser der Systemuhr des jeweiligen PCs.

Um die einzelnen Daten der Objektinstanz von `Date`, also Tag, Monat, Jahr usw. anzusprechen, stehen entsprechende Methoden zur Verfügung. Diese Methoden, z.B. `getDate()` oder `getHours()`, haben als Rückgabewert jeweils einen Datums/Uhrzeit-Bestandteil. So liefert `getDate()` beispielsweise den aktuellen Tag des Monats und `getHours()` die aktuelle Stundenzahl des Tages. Im Beispiel wird für jeden der benötigten Bestandteile eine Variable definiert. In der Variablen `Tag` wird beispielsweise durch Aufruf von `Jetzt.getDate()` der aktuelle Tag des Monats gespeichert.

Die Anweisung im Beispiel, die mit `NachVoll ...` beginnt, kann an dieser Stelle nicht näher erläutert werden. Die Anweisung ist eine Vorbereitung zur sauberen Formatierung der Uhrzeit.

**Beachten Sie:** Die einzelnen Methoden des im Beispiel verwendeten Objekts `Date`, wie zum Beispiel `getDate()`, werden bei der Referenz des Objekts `Date` in Kapitel 23 beschrieben.

### 18.3.3 Eigene Objekte definieren

Sie können eigene Objekte definieren, wenn Sie streng objektorientiert in JavaScript programmieren wollen. Um ein eigenes Objekt anzulegen, sind zwei Schritte nötig. Zuerst müssen Sie das Objekt selbst und seine Eigenschaften »deklarieren«. Im zweiten Schritt können Sie anschließend eine Instanz dieses Objekts definieren. Mit dieser Objekt-Instanz können Sie dann Programmprozeduren durchführen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Farbe(R, G, B) {
    this.R = R;
    this.G = G;
    this.B = B;
    this.hex="#";
}

function HintergrundWechseln() {
    var Hintergrund = new Farbe("E0","FF","E0");
    document.bgColor = Hintergrund.hex + Hintergrund.R + Hintergrund.G +
Hintergrund.B;
```

```

    }
    //-->
</script>
</head><body bgcolor="#FFFFFF">
<h1>Das eigene Farb-Objekt mit JavaScript</h1>
<a href="javascript:HintergrundWechseln()"><b>Hintergrundfarbe wechseln</b></a>
</body>
</html>

```

Um ein neues Objekt und seine Eigenschaften anzulegen, müssen Sie innerhalb eines JavaScript-Bereichs oder innerhalb einer separaten JavaScript-Datei eine eigene Funktion definieren, die so aussehen kann wie im Beispiel die Funktion `Farbe(...)`. Der Name, den Sie der Funktion geben (im Beispiel der Name `Farbe`), ist zugleich der Name des Objekts, das Sie mit dieser Funktion anlegen. Als Parameter, die die Funktion erwartet, notieren Sie die Eigenschaften, die Ihr Objekt haben soll. Die Parameternamen sind zugleich die Namen der Objekteigenschaften. Im Beispiel sind das die Eigenschaften `R`, `G` und `B`, denn es soll ein Objekt angelegt werden, das Rot-, Grün- und Blauwert einer Farbe speichern kann. `R`, `G` und `B` sind dabei übrigens selbst vergebene Namen. Innerhalb der Funktion notieren Sie alle Anweisungen so wie im Beispiel gezeigt: für jede Eigenschaft, die bei den Funktionsparametern festgelegt wurde, notieren Sie eine Anweisung, beginnend mit dem reservierten JavaScript-Wort `this`, gefolgt von einem Punkt und dem Parameternamen. Dahinter notieren Sie ein Gleichheitszeichen, und hinter dem Gleichheitszeichen nochmals den Parameternamen. Am Ende jeder Anweisung muss ein Strichpunkt stehen.

Natürlich können Sie für jedes Objekt auch feste (nicht durch Parameter beeinflusste) Eigenschaften definieren. Im Beispiel ist die Eigenschaft `hex` eine feste Eigenschaft, die für jede Instanz des Objektes existiert. Notieren Sie dazu hinter dem Eigenschaftsnamen den jeweiligen Wert der Eigenschaft.

Nachdem das Objekt angelegt ist, können Sie an anderen Stellen innerhalb Ihres JavaScripts Instanzen dieses Objekt definieren. Dies geschieht mit Hilfe einer Variablen und dem reservierten JavaScript Wort `new`. Im Beispiel wird eine zweite Funktion `HintergrundWechseln()` definiert. Darin wird zunächst eine Variable `Hintergrund` angelegt. Diese Variable soll Daten des angelegten Objekts `Farbe` enthalten. Dies geschieht durch ein Gleichheitszeichen hinter dem Variablennamen. Dahinter folgt das Wort `new` und dahinter, durch ein Leerzeichen getrennt, der Name der Funktion, mit der das gleichnamige Objekt angelegt wurde, im Beispiel `Farbe`. Als Parameter werden dieser Funktion irgendwelche brauchbaren Werte übergeben, im Beispiel `"33"`, `"99"` und `"C0"` (typische hexadezimale Farbwerte, wie sie in HTML zum Einsatz kommen).

Anschließend ist die Variable `Hintergrund` an das Farbojekt gebunden, und über sie lassen sich die Eigenschaften des Objekts setzen oder abfragen. Im Beispiel steht die Anweisung:

```
document.bgColor = Hintergrund.hex + Hintergrund.R + Hintergrund.G + Hintergrund.B;
```

Damit wird die Hintergrundfarbe der aktuell angezeigten Seite verändert. Normalerweise könnte man `document.bgColor` so etwas zuweisen wie `"#C0C0EE"`. Auch im Beispiel wird ein Farbwert dieser Art zugewiesen, jedoch zusammengesetzt aus den Einzeleigenschaften des Farbojekts, das in der Instanz `Hintergrund` gespeichert ist. Die Funktion `HintergrundWechseln()` wird aufgerufen, wenn der Anwender auf den Verweis klickt, der in der HTML-Datei enthalten ist.

### 18.3.4 Eigenschaften von Objekten

Objekte können Eigenschaften haben. Ein selbst definiertes Objekt »Mensch« könnte zum Beispiel die Eigenschaften Name, Alter, Geschlecht und Muttersprache haben. Vordefinierte JavaScript-Objekte haben ebenfalls Eigenschaften. So hat das Objekt `Math` zum Beispiel eine Eigenschaft »PI« (`Math.PI`). Auf diese Weise lässt sich mit der mathematischen Konstante PI rechnen, ohne deren genauen Wert zu kennen.

Eigenschaften von Objekten können Sie innerhalb Ihres JavaScript-Codes jederzeit auslesen, und in vielen Fällen können Sie die Werte von Eigenschaften auch ändern. So können Sie beispielsweise dem im Browser-Fenster angezeigten Dokument einen neuen, gültigen URI zuweisen. Dadurch bewirken Sie, dass der WWW-Browser einen Sprung zu der zugewiesenen Adresse ausführt, genau so, wie wenn der Anwender auf einen entsprechenden Verweis klicken würde. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<h1>Die Seite, die Sie genau erkennt</h1>
<script type="text/javascript">
<!--
var BrowserName = navigator.appName;
var BrowserVersion = navigator.appVersion;
document.write("<p>Ah ja, Sie verwenden also den <b>" + BrowserName + "</b>, und
zwar in der Version <b>" + BrowserVersion + "</b></p>");
//-->
</script>
</body></html>
```



18.4: Das Objekt `navigator` liefert Details über die Browser-Version und das Betriebssystem des Computers.

Im Beispiel werden innerhalb eines JavaScript-Bereichs zwei Eigenschaften des vordefinierten JavaScript-Objekts `navigator` in zwei entsprechenden Variablen namens `BrowserName` und `BrowserVersion` gespeichert. Das `navigator`-Objekt stellt Ihnen über seine Eigenschaften verschiedene Informationen über den verwendeten WWW-Browser des Anwenders zur Verfügung. Im obigen Beispiel werden die Eigenschaften des Browser-Namens (gespeichert in der Objekteigenschaft `navigator.appName`) und der Browser-Version (gespeichert in der Objekteigenschaft `navigator.appVersion`) ermittelt. Anschließend werden die ermittelten Daten mit `document.write()` dynamisch ins Browser-Fenster geschrieben.

Objekteigenschaften sprechen Sie an, indem Sie zuerst den Namen des Objekts notieren, dahinter einen Punkt, und dahinter den Namen der Eigenschaft. Dabei sind keine Leerzeichen erlaubt.

### 18.3.5 Objekt-Methoden

Objekte können Methoden haben. Das sind Funktionen, die Aktionen ausführen, aber im Gegensatz zu allein stehenden Funktionen an ein bestimmtes Objekt gebunden sind. Viele vordefinierte JavaScript-Objekte haben Methoden. So gibt es zum Beispiel das vordefinierte JavaScript-Objekt `history`, in dem die bereits besuchten URIs eines Browser-Fensters gespeichert sind. Dazu gibt es eine Methode `history.back()`, mit der Sie in JavaScript einen Rücksprung zu einer bereits besuchten Adresse erzwingen können. Ein Beispiel:

```
<html>
<head>
<title>Geh wo hin</title>
</head>
<body>
<a href="javascript:history.back();"><b>Geh hin wo Du herkommst</b></a>
</body>
</html>
```

Das Beispiel enthält einen Verweis mit einer speziellen Syntax. Diese Syntax erlaubt Ihnen, beim Anklicken des Verweises JavaScript-Code aufzurufen. Im Beispiel ist das ein Aufruf der Methode `back()` des Objekts `history`.

Objektmethoden sprechen Sie an, indem Sie zuerst den Namen des Objekts notieren, dahinter einen Punkt, dahinter den Namen der Methode, und dahinter eine öffnende und eine schließende Klammer. Dabei sind keine Leerzeichen erlaubt. Einige Methoden können auch Parameter beim Aufruf erwarten. Diese Parameter müssen Sie dann zwischen der öffnenden und der schließenden Klammer übergeben.

**Beachten Sie:** Mehr zu der Aufrufsyntax bei Verweisen erfahren Sie in Kapitel 18.10 über Event-Handler.

### 18.3.6 Abfragen, ob ein Objekt existiert

Es gibt viele JavaScript-Objekte, und es ist oft schwer durchschaubar, welcher Browser in welcher Version welches Objekt interpretiert. Objekte, die nicht interpretierbar sind, führen in den Browsern zu Fehlermeldungen. So kann es vorkommen, dass ein Script, das mit Netscape 4.7 oder Internet Explorer 5.0 wunderbar funktioniert, bei Netscape 4.0 oder Internet Explorer 4.0 zu Fehlermeldungen führt, weil ein Objekt nicht bekannt ist. Deshalb gibt es eine Möglichkeit, den Aufruf von Eigenschaften und Methoden eines Objekts von einer Abfrage abhängig zu machen, die herausfindet ob das Objekt dem Browser überhaupt bekannt ist. Ein Beispiel:

```
<html>
<head><title>Test</title>
</head>
<body>
```

```

<h1>Objekt-Abfrage</h1>
<script type="text/javascript">
<!--
if(! document.images)
  document.write("<p>Schade, Ihr Browser kennt das image-Objekt nicht - Sie
verpassen was!<br>");
else
  document.write("<p>Gut, Ihr Browser kennt das image-Objekt!<br>");
if(document.all)
  document.write("Ach, und Ihr Browser ist von Microsoft ;-)<\>p>");
else
  document.write("Ach, und Ihr Browser ist ja gar nicht von Microsoft ;-)<\>p>");
//-->
</script>
</body>
</html>

```

Mit einer bedingten Anweisung `if(...)` können Sie überprüfen, ob ein Objekt verfügbar ist. Als Ausdruck für die Bedingung notieren Sie innerhalb der Klammern hinter `if` einfach das Objekt, nach dem Sie fragen. Wenn das Objekt verfügbar ist, wird intern der Wert `true` (*wahr*) zurückgegeben. In dem Zweig der `if-else`-Abfrage, die diesen Fall verarbeitet, können Sie dann loslegen und zum Beispiel Befehle verwenden, die dieses Objekt verwenden. In dem anderen Zweig der `if-else`-Abfrage können Sie dann beispielsweise eine eigene Meldung ausgeben, dass dieses JavaScript leider Befehle verwendet, die der Browser des Anwenders nicht interpretiert.

Im obigen Beispiel werden mit `document.write` unterschiedliche Sätze ins Browser-Fenster geschrieben. Dazu wird zunächst gefragt, ob das Unterobjekt `document.images` **nicht** verfügbar ist (die verneinte Abfrage ergibt sich durch das Ausrufezeichen vor dem Objekt). Ist das Objekt also nicht verfügbar, dann wird im Beispiel ausgegeben, dass dies sehr schade sei usw. Im anderen Fall, also wenn das Objekt verfügbar ist, wird eine positive Reaktion ausgegeben.

Im zweiten Teil des Beispiels wird die Verfügbarkeit des Unterobjekts `document.all` abgefragt, das nur dem MS Internet Explorer bekannt ist. Auch dabei werden im Beispiel wieder je nach Verfügbarkeit zwei unterschiedliche Sätze ausgegeben. An diesem Beispiel ist auch ersichtlich, dass die Abfrage der Verfügbarkeit von Objekten auch zur Browser-Groberkennung dienen kann – vorausgesetzt, man kennt sich bei den vordefinierten JavaScript-Objekten aus und weiß, welche Objekte nur von bestimmten Browsern interpretiert werden.

### 18.3.7 Mehrere Anweisungen mit einem Objekt ausführen (with)

Hin und wieder kommt es vor, dass Sie mehrere Anweisungen in Folge notieren, die alle mit dem gleichen Objekt arbeiten. Für diesen Zweck können Sie, wenn Sie wollen, eine verkürzte, spezielle Schreibweise verwenden. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">

```

```

<!--
with(document) {
  open();
  write("Diese Seite hat den Titel " + title);
  close();
}
//-->
</script>
</body></html>

```

Mit `with(Objektnamen)` leiten Sie eine solche Schreibweise ein (*with* = *mit*). Da normalerweise mehrere Anweisungen folgen, die dieses Objekt verwenden, müssen Sie all diese Anweisungen in geschweifte Klammern einschließen.

Im obigen Beispiel wird mit dem `document`-Objekt gearbeitet. Innerhalb der geschweiften Klammern werden die Methoden `open()`, `write()` und `close()` sowie die Eigenschaft `title` verwendet. Sie alle gehören zum `document`-Objekt. Normalerweise müssten Sie notieren: `document.open()` oder `document.title`. Durch die spezielle Syntax mit `with(document)` entfällt dies.

**Beachten Sie:** Es ist auch möglich, untergeordnete Objekte auf diese Weise anzusprechen, z.B.:

```

with(document.MeinFormular.Feld_1)
  value = Stefan

```

Achten Sie darauf, dass die aufgerufene Eigenschaft auch tatsächlich existiert. Wenn sie fehlt, erhalten Sie eine JavaScript-Fehlermeldung.

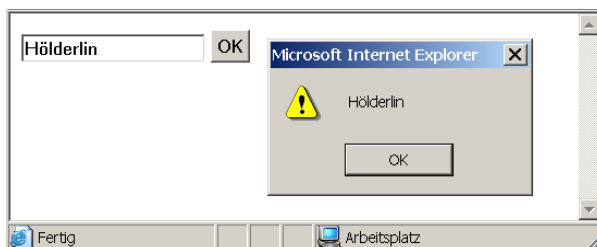
### 18.3.8 Auf aktuelles Objekt Bezug nehmen (this)

Es gibt Fälle, in denen es eindeutig ist, auf welches Objekt sich ein Befehl bezieht. In solchen Fällen können Sie eine verkürzte Schreibweise benutzen. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<form name="Eingabe">
<input type="text" name="Feld">
<input type="button" value="OK" onClick="alert(this.form.Feld.value)">
</form>
</body></html>

```



18.5: Die Meldungsbox greift über `this` auf den Inhalt des Eingabefeldes zu.

Mit dem Schlüsselwort `this` können Sie auf ein aktuelles Objekt Bezug nehmen. Im Beispiel wird ein Formular mit einem Eingabefeld und einem Button definiert. Wenn der Anwender auf den Button klickt, tritt der Event-Handler `onClick=` in Aktion, der als Attribut in dem HTML-Tag für den Button notiert ist. Im Beispiel wird in einem Meldungsfenster der Wert ausgegeben, den der Anwender in dem Eingabefeld eingegeben hat. Normalerweise würde der Befehl so notiert:

```
alert(document.Eingabe.Feld.value)
```

Da der Befehl jedoch innerhalb des Formulars steht, auf das er sich bezieht, dürfen Sie auch schreiben:

```
alert(this.form.Feld.value)
```

Das Wort `form` ist dabei vom `forms`-Objekt abgeleitet.

## 18.4 Funktionen

### 18.4.1 Funktion definieren

Mit Hilfe von Funktionen können Sie eigene, in sich abgeschlossene JavaScript-Prozeduren programmieren, die Sie dann über den Aufruf der Funktion ausführen können. Dabei können Sie bestimmen, bei welchem Ereignis (zum Beispiel, wenn der Anwender einen Button anklickt) die Funktion aufgerufen und ihr Programmcode ausgeführt wird. JavaScript-Code, der nicht innerhalb einer Funktion steht, wird beim Einlesen der Datei vom WWW-Browser sofort ausgeführt.

Eine Funktion ist ein Anweisungsblock. Sie können eigene Funktionen innerhalb eines JavaScript-Bereichs oder in einer separaten JavaScript-Datei definieren. An erlaubten Stellen, z.B. innerhalb der einleitenden HTML-Tags `<body...>` und `<a href...>` oder in einem Formular-Tag wie `<input...>`, können Sie eine solche selbst definierte Funktion dann mit Hilfe eines Event-Handlers aufrufen. Oder Sie rufen eine Funktion innerhalb einer anderen Funktion auf.

Beispiel einer JavaScript-Funktion:

```
function PrimzahlCheck(Zahl) {
    var Grenzzahl = Zahl / 2;
    var Check = 1;
    for(var i = 2; i <= Grenzzahl; i++)
        if(Zahl % i == 0)
        {
            alert(Zahl + " ist keine Primzahl, weil teilbar durch " + i);
            Check = 0;
        }
    if(Check == 1)
        alert(Zahl + " ist eine Primzahl!");
}
```

Mit dem Schlüsselwort `function` leiten Sie die Definition einer Funktion ein. Dahinter folgt, durch ein Leerzeichen getrennt, ein frei wählbarer Funktionsname, im Beispiel: `PrimzahlCheck`.



Vergeben Sie einen Funktionsnamen, der das, was die Funktion leistet, ungefähr beschreibt. Beachten Sie dabei die Regeln für selbst vergebene Namen (Kapitel 18.1.3).

Unmittelbar hinter dem Funktionsnamen folgt eine öffnende Klammer. Wenn die Funktion keine Parameter erwarten soll, notieren Sie dahinter sofort wieder eine schließende Klammer. Wenn die Funktion Parameter übergeben bekommen soll, notieren Sie innerhalb der Klammer die Namen der Parameter. Die Namen der Parameter sind frei wählbar. Bei den Parameternamen gelten die gleichen Regeln wie beim Funktionsnamen. Mehrere Parameter werden durch Kommata voneinander getrennt. Im obigen Beispiel erwartet die Funktion `PrimzahlCheck` einen Parameter `Zahl`.

Der gesamte Inhalt der Funktion wird in **geschweifte Klammern** `{ }` eingeschlossen. Diese Klammern dürfen niemals fehlen!

Die Anweisungen innerhalb der Funktion können sehr unterschiedlicher Natur sein. Da können Sie z.B. Objekte manipulieren, übergebene Parameter verarbeiten und ändern, Berechnungen anstellen usw. Sie können innerhalb von Funktionen auch andere Funktionen aufrufen. Welche Anweisungen innerhalb einer Funktion stehen, hängt davon ab, was die Funktion leisten soll. Im obigen Beispiel wird ermittelt, ob die übergebene Zahl eine Primzahl ist. Wenn es keine ist, wird für jede Zahl, durch die sie teilbar ist, eine entsprechende Meldung ausgegeben. Wenn es eine Primzahl ist, wird am Ende ausgegeben, dass es sich um eine Primzahl handelt.

## 18.4.2 Funktion aufrufen

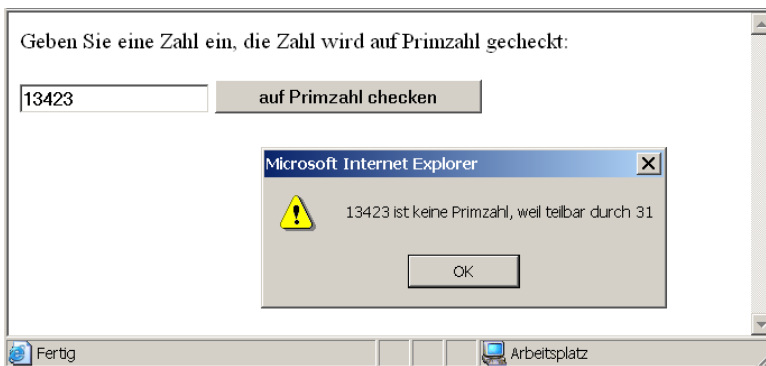
Sie können eine selbst definierte Funktion aufrufen, um den darin enthaltenen JavaScript-Code auszuführen. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function PrimzahlCheck(Zahl) {
    var Grenzzahl = Zahl / 2;
    var Check = 1;
    for(var i = 2; i <= Grenzzahl; i++)
        if(Zahl % i == 0)
        {
            alert(Zahl + " ist keine Primzahl, weil teilbar durch " + i);
            Check = 0;
        }
    if(Check == 1)
        alert(Zahl + " ist eine Primzahl!");
}
//-->
</script>
</head>
<body>
<form name="PrimzahlFormular" action="">
```

```

<p>Geben Sie eine Zahl ein, die Zahl wird auf Primzahl gecheckt:</p>
<input type="text" name="Eingabezahl">
<input type="button" value="auf Primzahl checken"
onClick="PrimzahlCheck(document.PrimzahlFormular.Eingabezahl.value)">
</form>
</body>
</html>

```



18.6: Die Primzahlenprüfung ist in eine Funktion ausgelagert.

Das obige Beispiel zeigt eine komplette HTML-Datei. Darin ist ein JavaScript-Bereich definiert, in dem wiederum die Funktion `PrimzahlCheck()` definiert ist. Im Dateikörper der HTML-Datei steht ein Formular mit einem Eingabefeld und einem Button. Im Eingabefeld kann der Anwender eine Zahl eingeben. Wenn er auf den Button klickt, wird die JavaScript-Funktion aufgerufen und ermittelt, ob es sich bei der Eingabe um eine Primzahl handelt. Dazu enthält der Button den Event-Handler `onClick=` (= beim Anklicken). Dahinter wird die JavaScript-Funktion mit der eingegebenen Zahl aufgerufen.

Die Funktion rufen Sie mit ihrem Funktionsnamen auf. Dahinter folgt die öffnende Klammer. Wenn die Funktion keine Parameter erwartet, notieren Sie hinter der öffnenden gleich eine schließende Klammer. Wenn die Funktion Parameter erwartet, müssen Sie für jeden Parameter einen erlaubten Wert übergeben. Im Beispiel erwartet die Funktion einen Parameter. Wenn Sie mehrere Parameter übergeben, trennen Sie diese durch Kommata.

### 18.4.3 Funktion mit Rückgabewert aufrufen

Eine Funktion kann einen ermittelten Wert an die aufrufende Instanz zurückgeben. Ein Beispiel:

```

<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function BruttoBetrag(Netto, Prozente) {

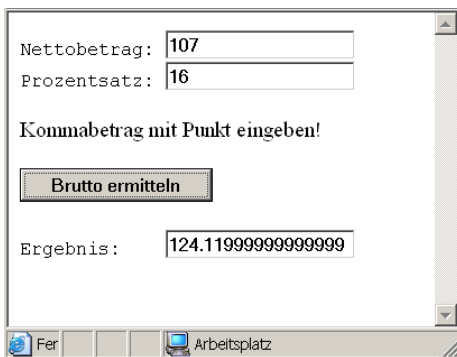
```

```

    var Ergebnis = Netto * (1 + (Prozente / 100));
    return Ergebnis;
}

function SchreibeBrutto(Betrag, Prozentsatz) {
    var Wert = BruttoBetrag(Betrag, Prozentsatz);
    document.BruttoForm.Ergebnisfeld.value = Wert;
}
//-->
</script>
</head>
<body>
<form name="BruttoForm" action="">
<pre>
Nettobetrag: <input type="text" name="NettoEingabe">
Prozentsatz: <input type="text" name="ProzentEingabe"></pre>
<p>Kommabetrag mit Punkt eingeben!</p>
<input type="button" value="Brutto ermitteln"
onClick="SchreibeBrutto(document.BruttoForm.NettoEingabe.value,document.BruttoForm.
ProzentEingabe.value)">
<pre>
Ergebnis:   <input type="text" name="Ergebnisfeld"></pre>
</form>
</body>
</html>

```



18.7: Die Prozentfunktion liefert den Bruttobetrag als Rückgabewert.

Das obige Beispiel zeigt eine HTML-Datei, in deren Dateikopf ein JavaScript-Bereich definiert ist, in dem wiederum die zwei Funktionen `BruttoBetrag(...)` und `SchreibeBrutto(...)` definiert sind. Im Dateikörper der HTML-Datei steht ein Formular mit je einem Eingabefeld für einen Nettobetrag und einen Prozentwert. Wenn der Anwender auf den darunter definierten Button klickt, wird die Funktion `SchreibeBrutto(...)` aufgerufen. Diese wiederum ruft die Funktion `BruttoBetrag(...)` auf.

Da die Funktion `BruttoBetrag(...)` Ihr errechnetes Ergebnis an die aufrufende Instanz zurückgibt, wird in `SchreibeBrutto(...)` eine Variable mit dem Namen `Wert` definiert, die diesen Rückgabewert speichert. Das Ergebnis, das in dieser Variablen gespeichert ist, schreibt die Funktion schließlich in ein Feld, das innerhalb des Formulars eigens für diesen Zweck eingerichtet wurde: das Eingabefeld mit dem Namen `Ergebnisfeld`.

#### 18.4.4 Vordefinierte JavaScript-Funktionen

Es gibt ein paar Funktionen, die bereits in JavaScript integriert sind. Solche Funktionen können Sie aufrufen, ohne sie selbst zu definieren. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
function Rechne(Operation) {
    var Ergebnis = eval(Operation);
    alert("Ergebnis: " + Ergebnis);
}
//-->
</script>
</head>
<body>
<form name="Formular">
<p>Geben Sie eine Rechenaufgabe (z.B. 8*5) ein:</p>
<input type="text" name="Eingabe">
<input type="button" value="OK"
onClick="Rechne(document.Formular.Eingabe.value)">
</form>
</body>
</html>
```

Das obige Beispiel zeigt eine HTML-Datei mit einem JavaScript-Bereich, in dem wiederum eine Funktion `Rechne()` definiert ist. Innerhalb des Dateikörpers der HTML-Datei ist ein Formular mit einem Eingabefeld notiert. In dem Eingabefeld kann der Anwender eine Rechenaufgabe eingeben, z.B. `1 + 1` oder `(28.76 - 0.00001) * 7`. Beim Anklicken des Klick-Buttons wird die Funktion `Rechne()` aufgerufen. Sie erwartet als Parameter eine Rechenaufgabe. Deshalb wird ihr der Inhalt des Formulareingabefeldes beim Aufruf übergeben. Die Funktion `Rechne` bedient sich zur Berechnung des Ergebnisses der äußerst mächtigen vordefinierten Funktion `eval()` (*eval* = *evaluate* = *berechne*). Diese kann – unter anderem – Rechenoperationen als solche erkennen und ausrechnen. Das Rechenergebnis wird im Beispiel in einem Meldungsfenster ausgegeben.

**Beachten Sie:** Eine Übersicht aller vordefinierten JavaScript-Funktionen finden Sie im Abschnitt 20.33 über *Objektunabhängige Funktionen*.

## 18.5 Steuerzeichen und besondere Notationen

### 18.5.1 Steuerzeichen bei Zeichenketten

Bei Zeichenkettenvariablen gibt es die Möglichkeit, Steuerzeichen in den Variablenwert einzufügen. Ein Beispiel:

```
<script type="text/javascript">
<!--
  var Variable1 = "Hier erfolgt ein\nZeilenumbruch";
  var Variable2 = "Hier erfolgt ein\fWagenrücklauf";
  var Variable3 = "Hier erfolgt ein\bBackspace";
  var Variable4 = "Hier erfolgt ein\rDOS-Extra-Zeilenumbruch";
  var Variable5 = "Hier erfolgt ein\tTabulator";
  var Variable6 = "Hier erfolgt ein\"Anführungszeichen";
// -->
</script>
```

Steuerzeichen dieser Art werden durch das Zeichen `\` eingeleitet. Dahinter folgt ein Buchstabe, der das Steuerzeichen markiert.

- Die Zeichenfolge `\n` ist z.B. in `alert`-Meldungen sinnvoll, um innerhalb des auszugebenden Textes einen Zeilenumbruch einzufügen.
- Die Zeichenfolge `\t` ist z.B. sinnvoll, um etwa innerhalb einer `alert(...)`-Meldung tabellarische Information auszugeben.
- Die Zeichenfolge `\r` sollten Sie zusätzlich zu dem Steuerzeichen `\n` notieren, wenn `\n` alleine nicht funktioniert.
- Die Zeichenfolge `\"` müssen Sie notieren, wenn Sie innerhalb einer Zeichenkette ein Anführungszeichen verwenden möchten (Maskierung des Zeichens, das die Zeichenkette einschließt).

### 18.5.2 Notation numerischer Werte

Sie können Zahlen ganz normal notieren. Beachten Sie dabei nur, dass bei Kommazahlen anstelle eines Kommas ein Punkt verwendet werden muss. So wird die Zahl Pi beispielsweise als 3.1415 notiert. Für sehr hohe und sehr niedrige Zahlen und für komplexe Kommazahlen gibt es daneben aber noch andere Notationsmöglichkeiten. Ein Beispiel:

```
<script language="JavaScript">
<!--
  var a = 1E1;
  var b = 1.2345E4;
  var c = 2e-3;
// -->
</script>
```

Mit `e` oder `E` bestimmen Sie die Zehnerpotenz bzw. die Anzahl Nullen, die hinter der Zahl vor dem `e` bzw. `E` stehen.

- Die erste Zahl im Beispiel, 1E1, ist eine 1 mit einer 0 dahinter, also 10.
- Die zweite Zahl im Beispiel, 1.2345E4, ist eine andere Schreibweise für 12345. Der Dezimalpunkt wird also einfach um so viele Stellen nach rechts verschoben, wie hinter dem E-Zeichen angegeben.
- Die dritte Zahl im Beispiel, 2e-3, ist eine andere Schreibweise für 0.002. Der Dezimalpunkt wird also einfach um so viele Stellen nach links verschoben, wie hinter dem E-Zeichen angegeben. Diese umgekehrte Richtung wird durch das Minuszeichen bewirkt, das hinter dem e folgt.

## 18.6 Operatoren

### 18.6.1 Zuweisungsoperator

Sie können zum Beispiel einer Variablen einen Wert zuweisen. Der Zuweisungsoperator dafür ist ein Gleichheitszeichen. Ein Beispiel:

```
<script type="text/javascript">
<!--
    var SinnDesLebens = 42;
// -->
</script>
```

Im Beispiel wird eine Variable namens `SinnDesLebens` definiert. Der Variablen wird mit dem Zuweisungsoperator `=` der Wert 42 (was sonst?) zugewiesen.

### 18.6.2 Vergleichsoperatoren

Vergleichsoperatoren brauchen Sie, wenn Sie zwei Werte vergleichen wollen, z.B. den aktuellen Inhalt einer Variablen mit einem fixen Wert. Vor allem bei bedingten Anweisungen und Schleifen kommt das vor. Ein Beispiel:

```
<script language="JavaScript" type="text/javascript">
<!--
    var SinnDesLebens=42;
    var Alter=8;

    if(SinnDesLebens == 42) alert(1);
    if(SinnDesLebens != 42) alert(0);
    if(SinnDesLebens > 42) alert(0);
    if(SinnDesLebens < 42) alert(0);
    if(Alter >= 18) alert("SIE duerfen das hier sehen!");
    if(Alter <= 17) alert("SIE duerfen das hier NICHT sehen!");
// -->
</script>
```

- Um abzufragen, ob zwei Werte gleich sind, notieren Sie zwei Gleichheitszeichen `==` nebeneinander.
- Um abzufragen, ob zwei Werte unterschiedlich sind, notieren Sie zwischen beiden Werten die Zeichen `!=`.
- Um abzufragen, ob ein Wert größer oder gleich ist als ein anderer, notieren Sie die Zeichen `>=`.
- Um abzufragen, ob ein Wert in jedem Fall größer ist als ein anderer, notieren Sie das Zeichen `>`.
- Um abzufragen, ob ein Wert kleiner oder gleich ist als ein anderer, notieren Sie die Zeichen `<=`.
- Um abzufragen, ob ein Wert in jedem Fall kleiner ist als ein anderer, notieren Sie das Zeichen `<`.

Nähere Information zu der If-Abfrage erhalten Sie im Abschnitt 18.7 über bedingte Anweisungen. Ab der JavaScript-Version 1.2 können Sie Gleichheit/Ungleichheit nicht nur mit `==` bzw. `!=`, sondern auch mit `===` bzw. `!==` abprüfen. In diesem Fall werden die Werte zusätzlich auf ihren Variablentyp hin überprüft. So wird z.B. die Anweisung `if (SinnDesLebens === "42") alert(1)` nicht ausgeführt. Berücksichtigen Sie bei der Verwendung dieser Operatoren, dass ältere Browser darauf mit einer Fehlermeldung reagieren.

### 18.6.3 Berechnungsoperatoren

Um mit numerischen Werten Berechnungen durchzuführen, brauchen Sie Berechnungsoperatoren. Ein Beispiel:

```
<script type="text/javascript">
<!--
  var Zwei = 1 + 1;
  var GarNix = 1 - 1;
  var AuchNix = 81 / 3 - 27;
  var WenigerAlsNix = 81 / (3 - 27);
  var SinnDesLebens = 6 * 7;
  var MachtAuchSinn = 84 / 2;
  var x = Jahr % 4;
  if(x == 0)
    Schaltjahr = true;

  /* Besondere Notationen: */

  var Zahl;
  Zahl+=3;
  Zahl++;
  Zahl-=2;
  Zahl--;
  Zahl*=4;
  Zahl/=3;
// -->
</script>
```

Mathematische Operatoren notieren Sie mit den dafür üblichen Zeichen. Mit + notieren Sie eine Addition, mit - eine Subtraktion, mit \* eine Multiplikation, mit / eine Division. Eine Besonderheit stellt der Operator % dar. Damit wird eine so genannte Modulo-Division durchgeführt. Bei einer Modulo-Division werden zwei Werte dividiert. Das Ergebnis ist jedoch im Gegensatz zur normalen Division nur der Restwert der Division. Wenn Sie z.B.  $13 \% 5$  notieren, erhalten Sie als Ergebnis 3, weil 13 geteilt durch 5 gleich 2 Rest 3 ergibt. Diese 3 ist es, die als Ergebnis einer Modulo-Division herauskommt.

Sie können mehrere Operationen in Reihe notieren. Dabei gilt die übliche »Punkt-vor-Strich-Regel«. Wenn Sie eine andere Regel erzwingen wollen, müssen Sie Klammern verwenden, so wie im vierten der obigen Beispiele.

Die besonderen Notationen, die in den obigen Beispielen vorkommen, können Sie verwenden, wenn Sie Additionen oder Subtraktionen abkürzen wollen:

- $\text{Zahl} += 3$ ; ist eine Abkürzung für  $\text{Zahl} = \text{Zahl} + 3$ ;
- $\text{Zahl} ++$ ; ist eine Abkürzung für  $\text{Zahl} = \text{Zahl} + 1$ ;
- $\text{Zahl} -= 2$ ; ist eine Abkürzung für  $\text{Zahl} = \text{Zahl} - 2$ ;
- $\text{Zahl} --$ ; ist eine Abkürzung für  $\text{Zahl} = \text{Zahl} - 1$ ;

Der Operator ++ wird auch als Inkrementationsoperator bezeichnet, der Operator -- als Dekrementationsoperator.

## 18.6.4 Logische Operatoren

Logische Operatoren brauchen Sie, wenn Sie komplexere Bedingungen für bedingte Anweisungen oder Schleifen formulieren wollen. Ein Beispiel:

```
<script type="text/javascript">
<!--
  var PLZ=81000;
  var x=20, y=8;

  if(PLZ >= 80000 && PLZ <= 82000)
    alert("Sie wohnen wohl in Muenchen oder Umgebung!")

  if(x > 100 || y == 0)
    break;
// -->
</script>
```

Mit dem logischen Operator && verknüpfen Sie zwei oder mehrere Bedingungen durch »und«, d.h. beide bzw. alle Bedingungen müssen erfüllt sein, damit die gesamte Bedingung erfüllt ist. Mit dem logischen Operator || verknüpfen Sie zwei oder mehrere Bedingungen durch »oder«, d.h. es genügt, wenn eine der Bedingungen erfüllt ist, damit die gesamte Bedingung erfüllt ist.



### 18.6.5 Bit-Operatoren

Bit-Operatoren sind nur etwas für Profis. Um Bit-Operatoren richtig einzusetzen, müssen Sie viel von computerinternen Speichervorgängen verstehen. Deshalb werden die Bit-Operatoren hier nur kurz erwähnt.

| Operator | Wirkung                                                   |
|----------|-----------------------------------------------------------|
| >>       | verschiebt Bits nach rechts                               |
| <<       | verschiebt Bits nach links                                |
| &        | definiert in einer Bitmaske eine Und-Bedingung            |
|          | definiert in einer Bitmaske eine inklusive Oder-Bedingung |
| ^        | definiert in einer Bitmaske eine exklusive Oder-Bedingung |

### 18.6.6 Operator zur Zeichenkettenverknüpfung

Mit einem einfachen Pluszeichen + können Sie eine Zeichenkette an eine andere anhängen. Ein Beispiel:

```
<script language="JavaScript" type="text/javascript">
<!--
var Vorname = "Stefan "
var Zuname = "Muenz"
var Name = Vorname + Zuname + ", der Autor dieses Dokuments"
// -->
</script>
```

### 18.6.7 Operatorenrangfolge

Unter den Operatoren von JavaScript gibt es eine festgelegte Rangordnung. Wenn Sie komplexe Rechenoperationen durchführen oder mehrere Bedingungen miteinander verknüpfen, gilt bei der internen Auflösung solcher komplexen Ausdrücke die folgende Rangordnung:

1. Rangstufe , (Aneinanderreihung)
2. Rangstufe = += -= << >>= &= ^= |=
3. Rangstufe ?: (Entweder-Oder-Bedingung)
4. Rangstufe | |
5. Rangstufe &&
6. Rangstufe |
7. Rangstufe ^
8. Rangstufe &
9. Rangstufe == === != !==

- 10. Rangstufe      < <= > >=
- 11. Rangstufe      << >> >>>
- 12. Rangstufe      + -
- 13. Rangstufe      \* / %
- 14. Rangstufe      ! ~ - ++ --
- 15. Rangstufe      ( ) [ ] . (Klammerung und Vektoren)

Mit Hilfe von Klammern, die absichtlich die unterste Rangstufe in der Prioritätshierarchie darstellen, können Sie die Rangfolge bei den Operatoren beeinflussen und Ausdrücke so bewerten, wie Sie es wünschen. Ein Beispiel:

```
<script type="text/javascript">
<!--
var OffizielleStatistik = 3.29 * 3 + 4.71;
var MeineStatistik = 3.29 * (3 + 4.71);
// -->
</script>
```

Das Beispiel zeigt, wie Sie durch Setzen von Klammern das Ergebnis einer Rechenoperation beeinflussen können.

### 18.6.8 Operator zur Typenbestimmung

Methoden vordefinierter Objekte sind an feste Variablentypen gebunden. Bevor Sie diese verwenden können, sollten Sie im Zweifelsfall den Typ der Variable prüfen. Mit dem Operator `typeof` steht Ihnen ein Werkzeug zur Typenüberprüfung zur Verfügung. Ein Beispiel:

```
<script type="text/javascript">
<!--
var Zahl=2505;
alert(typeof Zahl);
// -->
</script>
```

Der Operator `typeof` wertet den nachfolgenden Operanden hinsichtlich seines Typs aus. Der Rückgabewert dieses Operators ist ein String, den Sie abfragen können. Anweisungen müssen in Klammern eingeschlossen werden, z.B. `typeof(Zahl=Zahl+2)`. Mögliche Rückgabewerte sind:

| Variablentyp | Beschreibung          |
|--------------|-----------------------|
| boolean      | Ja/Nein-Variable      |
| string       | Zeichenkettenvariable |
| number       | numerische Variable   |
| function     | Funktion              |
| object       | Objekt                |
| undefined    | unbestimmter Typ      |

### 18.6.9 void-Operator

Der Operator `void` wird dann verwendet, wenn eine Anweisung ausgeführt, aber keine Rückgabewerte erzeugt werden dürfen. Das ist z.B. dann der Fall, wenn Sie so genannte »Bookmarklets« erzeugen möchten. Ein Beispiel:

```
<script type="text/javascript">
<!--
var Zahl=2505;
void Zahl++;
// -->
</script>
```

Der Operator `void` führt die nach ihm notierte Anweisung aus. Der Operator ersetzt dabei die Rückgabewerte der Anweisung stets durch `undefined`. Sie können auch eine Anweisung wie z.B. `Zahl=Zahl+2` als Operand angeben. In diesem Fall schließen Sie die Anweisung in Klammern ein, also `void(Zahl=Zahl+2)`.

### 18.6.10 Operator zum Löschen von Objekten

Nicht mehr benötigte Objekte bzw. Objekteigenschaften werden mit dem Operator `delete` gelöscht. Sie können jedoch keine vordefinierten Objekte und Objekteigenschaften löschen. Der Operator gibt im Erfolgsfall den Wert `true` und bei Misserfolg den Wert `false` zurück. Ein Beispiel:

```
<script type="text/javascript">
<!--
SinnDesLebens = 42;
delete SinnDesLebens;
alert(typeof SinnDesLebens);
// -->
</script>
```

Im Beispiel wird der Variablen `SinnDesLebens` der Wert 42 zugewiesen. Anschließend wird diese Variable mit `delete` wieder gelöscht. Mit Hilfe des Operators `typeof` wird überprüft, ob das Löschen der Variablen erfolgreich ist.

**Beachten Sie:** In Netscape bis einschließlich Version 6 können nur Variablen und Objekte gelöscht werden, die **nicht** mit dem Schlüsselwort `var` deklariert wurden. Auf vordefinierte Objekte oder Objekteigenschaften wie z.B. `Math.PI` können Sie diesen Operator **nicht** anwenden.

## 18.7 Bedingte Anweisungen (if-else/switch)

### 18.7.1 Wenn-dann-Bedingungen mit "if"

Sie können die Ausführung von Anweisungen von Bedingungen abhängig machen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
```

```
function Geheim() {  
  var Passwort = "Traumtaenzer";  
  var Eingabe = window.prompt("Bitte geben Sie das Passwort ein","");  
  if(Eingabe != Passwort) {  
    alert("Falsches Passwort!");  
    Geheim();  
  }  
  else  
    document.location.href="geheim.htm";  
}  
// -->  
</script>  
</head><body onload="Geheim()">  
<h1>Hier kommen Sie nur mit Passwort rein ;-)</h1>  
</body></html>
```

Das Beispiel stellt eine einfache Passwortabfrage dar. Das Script ist in einer Funktion namens `Geheim()` notiert, die aufgerufen wird, sobald die HTML-Datei vollständig geladen ist. Dazu ist im einleitenden `<body>`-Tag der Event-Handler `onLoad` notiert. Innerhalb der Funktion fordert ein Dialogfenster (`window.prompt()`) den Anwender auf, das Passwort einzugeben. Der Rückgabewert von `window.prompt()`, das eingegebene Passwort, wird in der Variablen `Eingabe` gespeichert.

Mit `if(Eingabe != Passwort)` fragt das Script ab, ob die Eingabe anders lautet als der der Variablen `Passwort` zuvor zugewiesene Wert `Traumtaenzer`. Ist dies der Fall, sind also beide Werte nicht gleich, dann war die Eingabe falsch. In diesem Fall wird mit `alert()` eine entsprechende Meldung ausgegeben. Anschließend wird die Funktion `Geheim()` erneut aufgerufen, und der Anwender kann das Passwort versuchen neu einzugeben. Im anderen Fall (`else`), wenn `Eingabe` und `Passwort` den gleichen Wert haben, wird mit `document.location.href` eine andere Seite aufgerufen, nämlich die »geschützte« Seite.

Mit `if` leiten Sie eine Wenn-dann-Bedingung ein (*if = wenn*). Dahinter folgt, in Klammern stehend, die Formulierung der Bedingung. Um solche Bedingungen zu formulieren, brauchen Sie Vergleichsoperatoren und in den meisten Fällen auch Variablen. Für Fälle, in denen die Bedingung nicht erfüllt ist, können Sie einen »andernfalls«-Zweig definieren. Dies geschieht durch `else` (*else = sonst*).

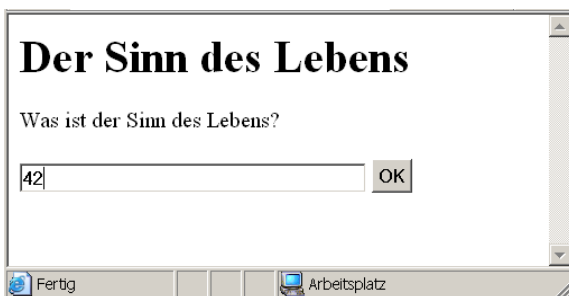
Der Else-Zweig ist nicht zwingend erforderlich. Wenn Sie mehr als eine Anweisung unterhalb und abhängig von `if` oder `else` notieren wollen, müssen Sie die Anweisungen in geschweifte Klammern einschließen (siehe auch den Abschnitt 18.1.2 über Anweisungsblöcke).

### 18.7.2 Einfache Entweder-oder-Abfrage

Für einfache Entweder-oder-Bedingungen gibt es eine spezielle Syntax, die Sie alternativ zur `if/else`-Anweisung verwenden können.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Antwort() {
  var Ergebnis = (document.Formular.Eingabe.value == "42") ? "RICHTIG!" : "FALSCH!";
  document.Formular.Eingabe.value =
    "Die Antwort ist " + Ergebnis;
}
// -->
</script>
</head><body>
<h1>Der Sinn des Lebens</h1>
<form name="Formular">
<p>Was ist der Sinn des Lebens?</p>
<input type="text" name="Eingabe" size="40">
<input type="button" value="OK" onClick="Antwort()">
</form>
</body></html>
```



18.8: Das Beispiel prüft den Inhalt eines Eingabefeldes auf Übereinstimmung mit einem vorgegebenen Wert.

Das Beispiel enthält eine JavaScript-Funktion namens `Antwort()`. Aufgerufen wird diese Funktion, wenn der Anwender in dem weiter unten notierten HTML-Formular auf den Klick-Button mit der Aufschrift *OK* klickt, und zwar mit dem Event-Handler `onClick`. Die Funktion prüft, ob der Wert des Eingabefeldes im Formular (`document.Formular.Eingabe.value`) den Wert 42 hat. Abhängig davon wird der Variablen `Ergebnis` entweder die Zeichenkette `RICHTIG!` oder `FALSCH!` zugewiesen. Anschließend wird in dem Textfeld des Formulars, das zur Eingabe diente, ein entsprechender Satz zusammengesetzt (siehe dazu auch *Operator für Zeichenkettenverknüpfung*, Kapitel 18.6.6).

Eine einfache Entweder-oder-Abfrage wird mit einer Bedingung eingeleitet. Die Bedingung muss in Klammern stehen, im Beispiel `(document.Formular.Eingabe.value == "42")`. Dahinter wird ein Fragezeichen notiert. Hinter dem Fragezeichen wird ein Wert angegeben, der aktuell ist, wenn die Bedingung erfüllt ist. Dahinter folgt ein Doppelpunkt, und dahinter ein Wert für den Fall, dass die Bedingung nicht erfüllt ist. Da es sich um Werte handelt, die für die Weiterver-

arbeitung nur sinnvoll sind, wenn sie in einer Variablen gespeichert werden, wird einer solchen Entweder-oder-Abfrage in der Regel eine Variable vorangestellt, im Beispiel die Variable `Ergebnis`. Der Variablen wird durch diese Art von Anweisung das Ergebnis der Entweder-oder-Abfrage zugewiesen.

Um Bedingungen zu formulieren, brauchen Sie die Vergleichsoperatoren aus Kapitel 18.6.2.

### 18.7.3 Fallunterscheidung mit »switch«

Mit `if` und `else` können Sie genau zwei Fälle unterscheiden. Wenn Sie feiner differenzieren, also zwischen mehreren Fällen unterscheiden wollen, können Sie zwar mehrere If-Abfragen hintereinander notieren, aber es gibt noch eine elegantere Möglichkeit: die Fallunterscheidung mit »switch«. Diese Syntax, die aus der Programmiersprache C entlehnt ist, gibt es in JavaScript aber erst seit der Sprachversion 1.2 – ältere Browser quittieren solche Anweisungen mit einer Fehlermeldung. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Eingabe = window.prompt("Geben Sie eine Zahl zwischen 1 und 4 ein:", "");
switch(Eingabe) {
  case "1":
    alert("Sie sind sehr bescheiden");
    break;
  case "2":
    alert("Sie sind ein aufrichtiger Zweibeiner");
    break;
  case "3":
    alert("Sie haben ein Dreirad gewonnen");
    break;
  case "4":
    alert("Gehen Sie auf allen Vieren und werden Sie bescheidener");
    break;
  default:
    alert("Sie bleiben leider dumm");
    break;
}
// -->
</script>
</body></html>
```

Mit `switch` leiten Sie eine Fallunterscheidung ein (*switch* = *Schalter*). Dahinter folgt, in runde Klammern eingeschlossen, eine Variable oder ein Ausdruck, für dessen aktuellen Wert Sie die Fallunterscheidung durchführen. Im Beispiel ist das die Variable mit dem Namen `Eingabe`. Diese Variable wird vor der Fallunterscheidung mit einem Wert versorgt, denn ein Dialogfenster (`window.prompt()`) mit einem Eingabefeld fordert den Anwender auf, eine Zahl zwischen 1 und 4 einzugeben. Der eingegebene Wert wird in `Eingabe` gespeichert.

Die einzelnen Fälle, die Sie abfragen möchten, werden innerhalb geschweiften Klammern notiert. Jeden einzelnen Fall leiten Sie mit `case` ein (*case* = *Fall*). Dahinter folgt die Angabe des Wertes, auf den Sie prüfen wollen. Die Anweisung `case "1"`: im obigen Beispiel bedeutet dann so viel wie: 'wenn die Variable Eingabe den Wert »1« hat'. Im Beispiel wird für jeden Fall eine individuelle Meldung ausgegeben.

Wichtig ist dabei auch das Wort `break` am Ende jedes Falls (*break* = *abbrechen*). Wenn Sie das Wort weglassen, werden nämlich alle nachfolgenden Fälle auch ausgeführt, aber das wollen Sie ja in der Regel nicht.

Für den Fall, dass keiner der definierten Fälle zutrifft, können Sie am Ende der Fallunterscheidung den Fall `default`: definieren. Die darunter stehenden Anweisungen werden ausgeführt, wenn keiner der anderen Fälle zutrifft.

## 18.8 Schleifen (while/for/do-while)

### 18.8.1 Schleifen mit »while«

Mit Hilfe von `while`-Schleifen können Sie Programmanweisungen so lange wiederholen, wie die Bedingung, die in der Schleife formuliert wird, erfüllt ist. Solche Schleifen eignen sich dann, wenn Sie nicht wissen, wie oft die Schleife durchlaufen werden soll. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Eingabe = "";
var Zaehler = 1;
while(Eingabe != "how to make love" && Zaehler <= 3) {
    Eingabe = window.prompt(Zaehler + ". Versuch: Was bedeutet 'HTML'?", "");
    Zaehler++;
}
if(Eingabe != "how to make love")
    document.write("<big>Lernen Sie erst mal HTML! ...</big>");
else
    document.write("<big>Fein, Sie haben verstanden worum es geht! ...</big>");
// -->
</script>
</body></html>
```

Das Beispiel bittet den Anwender in einer `while`-Schleife bis zu dreimal in einem Dialogfenster (`window.prompt()`), die Bedeutung der Abkürzung 'HTML' einzugeben. Die Schleife kann aus zwei Gründen beendet werden: Entweder der Anwender gibt die richtige Bedeutung der Abkürzung ein, oder die Variable `Zaehler`, die die Anzahl der Versuche mitzählt, hat einen Wert größer als 3. Wenn die Schleife beendet ist, steht also nicht fest, welche der beiden möglichen Ursachen sie beendet hat. Um das zu entscheiden, wird im Beispiel deshalb anschließend mit Hilfe einer `If`-Abfrage nochmals überprüft, ob die Schleife deshalb beendet wurde, weil die Ein-

gabe falsch war. Je nachdem, ob sie falsch oder richtig war, wird mit `document.write` ins Anzeigefenster des Browsers ein entsprechender Satz geschrieben.

Eine `while`-Schleife beginnt mit dem Wort `while` (*while* = *solange*). Dahinter folgt, in Klammern stehend, die Bedingung. Um eine Bedingung zu formulieren, brauchen Sie Vergleichsoperatoren. Der Inhalt der Schleife wird so lange wiederholt, wie die Schleifenbedingung wahr ist.

In der Regel enthält eine `while`-Schleife mehrere Anweisungen, die innerhalb der Schleife stehen. Notieren Sie alle Anweisungen innerhalb **geschweiffter** Klammern `{` und `}`, so wie im Beispiel (siehe auch den Abschnitt 18.1.2 über Anweisungsblöcke).

**Beachten Sie:** Achten Sie bei `while`-Schleifen immer darauf, dass es mindestens eine Möglichkeit gibt, um die Schleife nach einer angemessenen Zeit zu beenden. Andernfalls erzeugen Sie eine so genannte »Endlosschleife«, aus der der Anwender nur durch gewaltsames Beenden des WWW-Browsers herauskommt. Das ist besonders bei Online-Sitzungen im WWW sehr ärgerlich!

Um Endlosschleifen zu vermeiden, brauchen Sie irgendetwas, das irgendwann zu einem Ausweg aus der Schleife führt. Meistens werden zu diesem Zweck so genannte »Zähler« definiert, im Beispiel die Variable `Zaehler`. Diese Variable hat im Beispiel einen Anfangswert von 1 und wird innerhalb der Schleife bei jedem Durchgang mit der Anweisung `Zaehler++`; um 1 erhöht. Wenn im Beispiel der Zählerstand größer als 3 ist, wird abgebrochen. Weitere Möglichkeiten, um Schleifen abubrechen, werden im Abschnitt 18.8.4 beschrieben.

### 18.8.2 Schleifen mit »for«

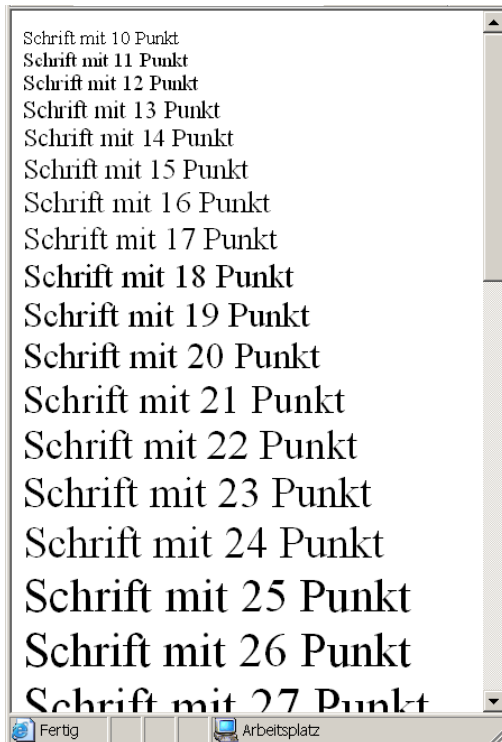
Die Schleifenbedingung einer `for`-Schleife sieht von vorneherein einen Zähler und eine Abbruchbedingung vor. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var Ausgabe = "";
  for(var i = 10; i <= 36; i++)
    Ausgabe = Ausgabe + "<span style=\"font-size:" + i + "pt\">Schrift mit " + i + "
Punkt</span><br>";
  document.write(Ausgabe);
// -->
</script>
</body></html>
```

Das Beispiel definiert eine Variable namens `Ausgabe`, die im Verlauf einer `for`-Schleife immer mehr Inhalt erhält und am Ende mit `document.write` ihren ganzen Inhalt ins Browser-Fenster schreibt. Die `for`-Schleife wird insgesamt 27-mal durchlaufen, nämlich so oft, wie der Zähler, der in der Variablen `i` definiert und mit dem Wert 10 initialisiert wird, kleiner oder gleich 36 ist, wobei er bei jedem Schleifendurchlauf um 1 erhöht wird (`i++`). Mit jedem Schleifendurchgang wird die Variable `Ausgabe` mit ihrem jeweils bisherigen Wert um etwas HTML-Code mit der CSS-Angabe `font-size` (Schriftgröße) erweitert. Der Wert, der `font-size` dabei zugewiesen



wird, ist jeweils der Wert von `i`. So entsteht der Effekt, dass CSS-Angaben von `font-size:10pt` bis `font-size:36pt` erzeugt werden. Das Ergebnis wird am Ende ausgegeben. Zum Verständnis der zusammengesetzten Teile bei Ausgabe siehe auch Operator für Zeichenkettenverknüpfung.



18.9: Die `for`-Schleife gibt denselben Text mit immer größer werdender Schrift aus.

Eine `for`-Schleife beginnt mit dem Wort `for`. Dahinter wird, in Klammern stehend, die Schleifenbedingung formuliert. Bei der `for`-Schleife gilt dabei eine feste Syntax. Innerhalb der Schleifenbedingung werden drei Anweisungen notiert. In der ersten Anweisung wird ein Schleifenzähler definiert und initialisiert. Im Beispiel wird ein Zähler `i` definiert und mit dem Wert 10 initialisiert. Die zweite Anweisung enthält die Bedingung, ab der die Schleife beendet wird. Dazu brauchen Sie Vergleichsoperatoren. In der dritten Anweisung wird der Schleifenzähler so verändert, dass er irgendwann die in der zweiten Anweisung notierte Bedingung erfüllt. Im Beispiel wird der Zähler mit `i++` bei jedem Schleifendurchgang um 1 erhöht. An dieser Stelle könnte aber auch so etwas stehen wie `i=i+10` (bei jedem Schleifendurchgang um 10 erhöhen).

Eine spezielle Abart der `for`-Schleife ist die `for-in`-Schleife. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var Ausgabe = "";
  for (var Eigenschaft in document)
```

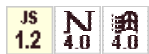
```

    Ausgabe = Ausgabe + "document." + Eigenschaft + ": " + document[Eigenschaft] +
"<br>";
    document.write("<h1>Eigenschaften des Objekts <i> document </i></h1>")
    document.write(Ausgabe);
// -->
</script>
</body></html>

```

Das Beispiel zeigt, wie Sie mit Hilfe einer *for-in*-Schleife einiges über die JavaScript-Fähigkeiten Ihres Browsers herausbekommen können. Im Beispiel werden die Eigenschaften des Objektes *document* ausgegeben. Mit jedem Schleifendurchgang wird die Variable *Ausgabe* um eine Objekteigenschaft erweitert. Den aktuellen Wert der Objekteigenschaft können Sie sich mit *Objektnamen[Eigenschaft]* ausgeben lassen. Die Schleife läuft so lange, wie es verfügbare Objekteigenschaften gibt – dies bedeutet das *for in*.

**Beachten Sie:** Wenn Sie mehr als eine Anweisung von einer *for*-Schleifenbedingung abhängig machen wollen, müssen Sie die Anweisungen wie üblich in geschweifte Klammern einschließen. Die *for-in*-Schleife können Sie in Opera nur auf selbst definierte Objekte und Variablen anwenden. Bei vordefinierten Objekten bleibt die Schleife wirkungslos. Der MS Internet Explorer kennt diese Schleifenart erst ab Version 4.01.



### Schleifen mit »do-while«

Die *do-while*-Schleife ist eine Variante der normalen *while*-Schleife. Der Unterschied zwischen beiden ist, dass bei der normalen *while*-Schleife **vor** dem Ausführen des Codes die Schleifenbedingung überprüft wird, während bei der *do-while*-Schleife **zuerst** der Code ausgeführt und erst danach die Schleifenbedingung überprüft wird. Auf diese Weise können Sie erzwingen, dass Anweisungen innerhalb der Schleife auf jeden Fall mindestens einmal ausgeführt werden, auch wenn sich die Schleifenbedingung gleich am Anfang als unwahr herausstellt. Ein Beispiel:

```

Einmal so:
<script type="text/javascript">
<!--
var x = 10;
do {
    document.write("<br>x * x = " + (x * x));
    x = x + 1;
}
while(x < 10);
// -->
</script>
Und einmal so:
<script type="text/javascript">
<!--
var x = 10;
while(x < 10) {
    document.write("<br>x * x = " + (x * x));
    x = x + 1;
}

```

```

}
// -->
</script>

```

Im Beispiel werden zwei kleine JavaScript-Bereiche definiert. In beiden Bereichen wird eine Variable `x` definiert und mit dem Wert 10 vorbelegt. Im ersten Bereich wird so lange das Quadrat von `x` (das bei jedem Schleifendurchlauf um 1 erhöht wird) geschrieben, wie `x` kleiner als 10 ist. Da `x` ja schon am Beginn den Wert 10 hat, ist die Abbruchbedingung eigentlich schon von vorneherein erfüllt. Trotzdem wird einmal das Quadrat von `x` ausgegeben, da die Überprüfung der Schleifenbedingung erst nach dem Ausführen der Anweisungen innerhalb der Schleife erfolgt.

Im zweiten Script-Bereich herrschen die gleichen Bedingungen, jedoch wird dort eine normale `while`-Schleife notiert. Da `x` von vorneherein nicht kleiner als 10 ist, werden die Anweisungen der `while`-Schleife kein einziges Mal ausgeführt. Die Überprüfung der Schleifenbedingung, die am Anfang stattfindet, verhindert dies.

### 18.8.3 Kontrolle innerhalb von Schleifen – `break` und `continue`

Schleifen sind »kritische Faktoren« innerhalb eines Scripts. Bei komplizierteren Aufgaben ist es manchmal nicht einfach, eine Schleife so zu programmieren, dass sie in jedem Fall irgendwann mal abgebrochen wird. Deshalb gibt es zusätzliche Befehle, um innerhalb einer Schleife das Geschehen zu kontrollieren. Ein Beispiel 1:

```

<script type="text/javascript">
<!--
var i = 0;
while (i < 6) {
  if (i == 3) break;
  i++;
  alert("i = " + i);
}
// -->
</script>

```

Mit `break` können Sie eine Schleife sofort beenden. Dazu müssen Sie innerhalb des Schleifenkörpers eine `if`-Abfrage notieren und abhängig davon das Wort `break` als Anweisung notieren. Im Beispiel bricht die Schleife bereits ab, wenn `i` den Wert 3 hat, obwohl laut Schleifenbedingung das Hochzählen bis 6 erlaubt.

Beispiel 2:

```

<script type="text/javascript">
<!--
var i = 0;
while (i < 6) {
  Ende:
  if (i == 3) {
    alert("Das war's, denn i ist gleich " + i);

```

```

        break Ende;
    }
    i++;
}
// -->
</script>

```

Dies ist eine besondere Variante von `break`. Sie können vor einer Abfrage, von der Sie eine `break`-Anweisung abhängig machen, ein »Label« notieren. Das ist ein selbst vergebener Name mit einem Doppelpunkt dahinter, im Beispiel `Ende:`. Hinter dem Wort `break` können Sie dann den Namen des Labels angeben. So stellen Sie im Beispiel sicher, dass sich die `break`-Anweisung auf jeden Fall auf jene `if`-Abfrage bezieht, in der abgefragt wird, ob `i` gleich 3 ist. In einfachen Fällen wie im obigen Beispiel ist das eigentlich überflüssig. Aber behalten Sie die Möglichkeit im Auge, falls Sie einmal verschachtelte `if`-Abfragen innerhalb von Schleifen programmieren und dort auch `break`-Anweisungen benutzen. Beachten Sie jedoch, dass diese Variante der `break`-Anweisung JavaScript 1.2 ist und bei älteren Browsern zu einer Fehlermeldung führt.

Beispiel 3:

```

<script type="text/javascript">
<!--
var i = 0, j = 0;
while (i < 6) {
    i++;
    if (i == 3) continue;
    j++;
}
alert("i ist gleich " + i + " und j ist gleich " + j);
</script>

```

Mit `continue` erzwingen Sie sofort den nächsten Schleifendurchlauf. Nachfolgende Anweisungen innerhalb der Schleife werden bei diesem Schleifendurchlauf nicht mehr ausgeführt. Im Beispiel werden zwei Zähler `i` und `j` bei jedem Schleifendurchlauf um 1 erhöht. Wenn `i` gleich 6 ist, wird die Schleife abgebrochen. Zwischendurch hat `i` auch mal den Wert 3. Dieser Fall wird mit einer `if`-Abfrage behandelt. Wenn `i` gleich 3 ist, wird sofort der nächste Schleifendurchgang gestartet. Die Anweisung `j++`; wird dadurch in diesem Schleifendurchlauf nicht mehr ausgeführt. Am Ende hat dadurch `i` den Wert 6 und `j` nur den Wert 5.

## 18.9 Reservierte Wörter

### 18.9.1 Übersicht reservierter Wörter

JavaScript enthält eine Reihe von Schlüsselwörtern mit bestimmter Bedeutung. Diese »reservierten Wörter« sollten Sie kennen, um nicht versehentlich gleichnamige Variablen oder Funktionen zu definieren, denn dies ist nicht erlaubt. Einige der reservierten Wörter der folgenden Liste sind bereits in Gebrauch, andere sind für den zukünftigen Sprachausbau von JavaScript geplant. Auch noch nicht benutzte reservierte Wörter dürfen Sie nicht als Variablen- oder Funktionsnamen verwenden.

| <i>Reserviertes Wort</i> | <i>Erläuterung</i>                                                                  |
|--------------------------|-------------------------------------------------------------------------------------|
| abstract                 | noch nicht verwendet                                                                |
| boolean                  | noch nicht verwendet                                                                |
| break                    | als Anweisung: Abbruch in Schleifen                                                 |
| byte                     | noch nicht verwendet                                                                |
| case                     | für Fallunterscheidungen                                                            |
| catch                    | als Anweisung: dient zur Fehlerbehandlung (ab JavaScript-Version 1.4)               |
| char                     | noch nicht verwendet                                                                |
| class                    | noch nicht verwendet                                                                |
| const                    | Dient zur Deklaration von Konstanten (ab JavaScript-Version 1.5)                    |
| continue                 | als Anweisung: Fortsetzung in Schleifen                                             |
| default                  | für Fallunterscheidungen                                                            |
| delete                   | Operator: Objekte und Objekteigenschaften löschen                                   |
| do                       | für Schleifen mit »do-while«                                                        |
| double                   | noch nicht verwendet                                                                |
| else                     | sonst-Fall in bedingten Anweisungen mit »if«                                        |
| export                   | als Anweisung: Objekte oder Funktionen für fremde Scripts ausführbar machen         |
| extends                  | noch nicht verwendet                                                                |
| false                    | rückgabewert von Funktionen: falsch                                                 |
| final                    | noch nicht verwendet                                                                |
| finally                  | noch nicht verwendet                                                                |
| float                    | noch nicht verwendet                                                                |
| for                      | für Schleifen mit »for«                                                             |
| function                 | für Funktionen                                                                      |
| goto                     | noch nicht verwendet                                                                |
| if                       | für bedingte Anweisungen mit »if«                                                   |
| implements               | noch nicht verwendet                                                                |
| in                       | für spezielle Schleifen mit »for«                                                   |
| instanceof               | prüft, ob eine Variable eine Instanz eines Objektes ist (ab JavaScript-Version 1.4) |
| int                      | noch nicht verwendet                                                                |
| long                     | noch nicht verwendet                                                                |
| native                   | noch nicht verwendet                                                                |
| new                      | Dient zum Definieren von Objekten                                                   |
| null                     | setzt ein Objekt auf null, ohne das Objekt selbst zu löschen                        |
| package                  | noch nicht verwendet                                                                |
| private                  | noch nicht verwendet                                                                |
| protected                | noch nicht verwendet                                                                |
| public                   | noch nicht verwendet                                                                |
| return                   | für Rückgabewerte in Funktionen                                                     |
| short                    | noch nicht verwendet                                                                |
| static                   | noch nicht verwendet                                                                |
| super                    | noch nicht verwendet                                                                |
| switch                   | für Fallunterscheidungen                                                            |
| synchronized             | noch nicht verwendet                                                                |
| this                     | für Bezug auf das aktuelle Objekt                                                   |

| Reserviertes Wort | Erläuterung                                                                         |
|-------------------|-------------------------------------------------------------------------------------|
| throw             | als Anweisung: nutzerdefinierte Ausnahme (ab JavaScript-Version 1.4)                |
| throws            | noch nicht verwendet                                                                |
| transient         | noch nicht verwendet                                                                |
| true              | Rückgabewert von Funktionen: wahr                                                   |
| try               | als Anweisung: testet eine Anweisung auf Ausführbarkeit (ab JavaScript-Version 1.4) |
| typeof            | Operator: liefert den Typ eines Elementes                                           |
| var               | für Variablendefinitionen                                                           |
| void              | Operator: unterdrückt Rückgabewerte                                                 |
| while             | für Schleifen mit »while«                                                           |
| with              | um mehrere Anweisungen mit einem Objekt durchzuführen                               |

## 18.10 Event-Handler

### Allgemeines zu Event-Handlern

Event-Handler (*Ereignis-Handler*) sind ein wichtiges Bindeglied zwischen HTML und JavaScript. Event-Handler werden meist in Form von Attributen in HTML-Tags notiert. Da es sich um Bestandteile handelt, die innerhalb von HTML vorkommen, hat das W3-Konsortium die Event-Handler mittlerweile auch in den HTML-Sprachstandard mit aufgenommen. Dort wird auch festgelegt, in welchen HTML-Tags welcher Event-Handler vorkommen darf. Das Problem dabei ist jedoch, dass die Praxis derzeit noch stark davon abweicht – zumindest bei Netscape 4.x. Der MS Internet Explorer dagegen interpretiert Event-Handler seit seiner Version 4.x weitgehend so universell, wie vom W3-Konsortium vorgesehen. Bei den Beschreibungen der Event-Handler auf dieser Seite wird jeweils versucht, auf die Problematik einzugehen. Letztendlich hilft aber nur: selber im Einzelfall mit mehreren verschiedenen Browsern testen und ausprobieren.

Event-Handler erkennen Sie daran, dass solche HTML-Attribute immer mit `on` beginnen, zum Beispiel `onClick=`. Hinter dem Gleichheitszeichen notieren Sie – in Anführungszeichen – eine JavaScript-Anweisung. Wenn Sie mehrere Anweisungen ausführen wollen, dann definieren Sie sich dazu am besten in einem JavaScript-Bereich eine Funktion und rufen hinter dem Gleichheitszeichen diese Funktion auf, also z.B. `onClick="Umrechnen()"`.

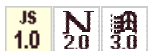
Jeder Event-Handler steht für ein bestimmtes Anwenderereignis, `onClick=` etwa für das Ereignis »Anwender hat mit der Maus geklickt«. Der Anzeigebereich des HTML-Elements, in dem der Event-Handler notiert ist, ist das auslösende Element. Wenn der Event-Handler `onClick=` beispielsweise in einem Formular-Button notiert wird, wird der damit verknüpfte JavaScript-Code nur dann ausgeführt, wenn der Mausklick im Anzeigebereich dieses Elements erfolgt. Das mag Ihnen jetzt selbstverständlich vorkommen. Ist es auch, solange es beispielsweise um Formular-Buttons geht. Aber nach dem erweiterten Modell von HTML 4.0 kann etwa auch ein HTML-Bereich, der mit `<div>...</div>` definiert wird, einen Event-Handler wie `onClick=` enthalten.

Es wurden nur solche Event-Handler aufgenommen, die auch tatsächlich in HTML-Tags vorkommen können und im HTML 4.0-Standard erwähnt sind (mit Ausnahme von `onAbort=` und `onError=`). Das sind weniger, als Netscape und der MS Internet Explorer kennen. Bei Netscape

kommt verwirrenderweise hinzu, dass einige Event-Handler zwar so bezeichnet werden, aber eigentlich gar nicht innerhalb von HTML-Tags vorkommen können. Es ist zu hoffen, dass es hierbei in Zukunft mehr Übereinstimmungen zwischen Sprachstandards und Browser-Implementierungen geben wird.

Im Folgenden finden Sie diese Event-Handler:

- `onAbort` (bei Abbruch)
- `onBlur` (beim Verlassen)
- `onChange` (bei erfolgter Änderung)
- `onClick` (beim Anklicken)
- `onDbClick` (bei doppeltem Anklicken)
- `onError` (im Fehlerfall)
- `onFocus` (beim Aktivieren)
- `onKeyDown` (bei gedrückter Taste)
- `onKeyPress` (bei gedrückt gehaltener Taste)
- `onKeyUp` (bei losgelassener Taste)
- `onLoad` (beim Laden einer Datei)
- `onMouseDown` (bei gedrückter Maustaste)
- `onMouseMove` (bei weiterbewegter Maus)
- `onMouseout` (beim Verlassen des Elements mit der Maus)
- `onmouseover` (beim Überfahren des Elements mit der Maus)
- `onMouseUp` (bei losgelassener Maustaste)
- `onReset` (beim Zurücksetzen des Formulars)
- `onSelect` (beim Selektieren von Text)
- `onSubmit` (beim Absenden des Formulars)
- `onUnload` (beim Verlassen der Datei)
- `javascript:` (bei Verweisen)



### ***onAbort (bei Abbruch)***

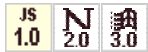
Ist für den Fall gedacht, dass ein Anwender im Browser den Stop-Button drückt, obwohl noch nicht alle Grafiken geladen wurden. Nach JavaScript (Netscape) erlaubt in folgendem HTML-Tag: `<img>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>

</body></html>
```

Im Beispiel wird eine Grafik in HTML referenziert. Für den Fall, dass der Anwender den Stop-Button im Browser drückt, bevor die Grafik ganz geladen ist, wird mit `alert()` eine Meldung ausgegeben.

**Beachten Sie:** Dieser Event-Handler gehört nicht zum HTML-Standard und wird von Netscape 6 und Opera 5 nicht interpretiert.



### *onBlur (beim Verlassen)*

Für den Fall, dass ein Element zuvor aktiviert war und der Anwender es jetzt verlässt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<body>` `<frameset>` `<input>` `<layer>` `<select>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<area>` `<button>` `<input>` `<label>` `<select>` `<textarea>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
Name: <input type="text" name="Eingabe" onBlur="CheckInhalt(this.value)"><br>
Name oder nichts eingeben und dann woanders hinklicken!
</form>
<script type="text/javascript">
document.Test.Eingabe.focus();
function CheckInhalt(Feld)
{
  if(Feld == "") {
    alert("Namensfeld muss einen Inhalt haben!");
    document.Test.Eingabe.focus();
    return false;
  }
}
</script>
</body></html>
```



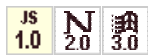
18.10: Verliert das Eingabefeld den Fokus, prüft das Script den Inhalt.

Im Beispiel wird ein Formular definiert, das ein Eingabefeld enthält. Unterhalb des Formulars ist ein JavaScript-Bereich notiert. Der Bereich wird deshalb unterhalb des Formulars definiert, weil



zu Beginn des Bereichs gleich eine Anweisung ausgeführt wird, die die Existenz des Formulars bereits voraussetzt. Diese Anweisung (`document.Test.Eingabe.focus();`) setzt den Cursor in das Eingabefeld. Dort soll der Anwender seinen Namen eingeben. Klickt er dann irgendwo anders hin, wird der Event-Handler `onBlur` aktiv, der im HTML-Tag des Eingabefeldes notiert ist. Dabei wird die Funktion `CheckInhalt()` aufgerufen, die ebenfalls in dem JavaScript-Bereich notiert ist. Diese Funktion fragt ab, ob die ihr übergebene Zeichenkette, der Inhalt des Namensfeldes, leer ist. Wenn ja, wird ein Meldungsfenster ausgegeben, und der Cursor wird wieder in das Feld positioniert.

**Beachten Sie:** Unter Opera 5.02 erzeugt die Verwendung dieses Beispiels eine unendliche Schleife, da die Bestätigung des Meldungsfensters ebenfalls den Event-Handler auslöst. Mit Opera 5.12 wurde dieses Problem behoben.

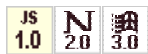


### ***onChange (bei erfolgter Änderung)***

Für den Fall, dass ein Element einen geänderten Wert erhalten hat. Nach HTML 4.0 und JavaScript 1.2 (Netscape) erlaubt in folgenden HTML-Tags: `<input>` `<select>` `<textarea>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<textarea rows="5" cols="40" onChange="alert(this.value)">Bearbeiten Sie diesen
Text hier und klicken Sie dann woanders hin! Oder lassen Sie ihn unbearbeitet und
klicken dann woanders hin!</textarea>
</body></html>
```

Im Beispiel wird ein Formular mit einem mehrzeiligen Eingabefeld definiert. Wenn der Anwender irgendetwas in das Feld eingibt und anschließend woanders hin klickt, wird der Event-Handler `onChange` aktiv, der im HTML-Tag des mehrzeiligen Eingabefeldes notiert ist. Im Beispiel wird einfach der aktuelle geänderte Inhalt des Feldes in einem Meldungsfenster ausgegeben.



### ***onClick (beim Anklicken)***

Für den Fall, dass der Anwender ein Element anklickt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<a>` `<area>` `<input>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input size="30" name="Ausgabe" readonly><br>
<input type="button" value="Letzter Update"
```

```
onClick="this.form.Ausgabe.value=document.lastModified">
</form>
</body></html>
```

Im Beispiel wird ein Formular mit einem Eingabefeld (das jedoch auf »readonly«, also nur Lesen gesetzt wird) und einem Button definiert. Der Button hat die Aufschrift »Letzter Update«. Beim Anklicken des Buttons wird der Event-Handler `onClick` aktiv, der im HTML-Tag des Buttons definiert ist. Im Beispiel wird daraufhin in das Eingabefeld der Zeitpunkt der letzten Änderung am Dokument geschrieben.



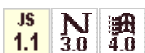
### ***onDbClick (bei doppeltem Anklicken)***

Für den Fall, dass der Anwender ein Element doppelt anklickt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<a>` `<area>` `<input>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Rechnen" action="">
Wert: <input size="10" name="Wert">
<input type="button" value="Doppelklick = hoch 2"
onDbClick="document.Rechnen.Wert.value=document.Rechnen.Wert.value*document.Rechne
n.Wert.value">
</form>
</body></html>
```

Im Beispiel wird ein Formular mit einem Eingabefeld und einem Button definiert. Im Button ist der Event-Handler `onDbClick` notiert. Der Button reagiert daher nur auf Doppelklick. Wenn der Anwender doppelt auf den Button klickt, wird von dem Wert, den er in dem Eingabefeld eingegeben hat, das Quadrat errechnet, und das Ergebnis wird wiederum in das Eingabefeld geschrieben.

**Beachten Sie:** Bei Netscape-Browsern unter Macintosh und bei Opera ist dieser Event-Handler nicht verfügbar!



### ***onError (im Fehlerfall)***

Eignet sich zum Abfangen von Fehlermeldungen und zum Ersetzen solcher Meldungen durch eigene. Beachten Sie jedoch, dass dadurch nicht die Fehler selbst beseitigt werden! `onError` ist vor allem zum Handling von Fehlern beim Laden von Grafiken gedacht. Nach JavaScript erlaubt in folgendem HTML-Tag: `<img>`.

Ein Beispiel:

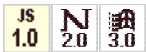
```
<html><head><title>Test</title>
</head><body>

</body></html>
```



18.11: Weil die Grafik nicht geladen werden kann, erscheint eine Fehlermeldung.

Im Beispiel wird in einer Grafikreferenz der Event-Handler `onError=` notiert. Er wird dann aktiv, wenn die Grafikdatei nicht existiert oder nicht angezeigt werden kann. Im Beispiel wird dann ein entsprechendes Meldungsfenster ausgegeben.



### ***onFocus (beim Aktivieren)***

Tritt ein, wenn der Anwender ein Element aktiviert. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<body>` `<frame>` `<input>` `<layer>` `<select>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<area>` `<button>` `<input>` `<label>` `<select>` `<textarea>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input size="30" onFocus="this.value='Hier Ihren Namen eingeben'"><br>
<input size="30" onFocus="this.value='Hier Ihren Wohnort eingeben'"><br>
<input size="30" onFocus="this.value='Hier Ihr Alter eingeben'"><br>
</form>
</body></html>
```

In dem Beispiel wird ein Formular definiert, das drei Eingabefelder enthält. Da die Felder unbeschriftet sind, hat der Anwender keine Ahnung, was er in die einzelnen Felder eingeben kann. Bewegt er den Cursor aus Neugier doch in eines der Eingabefelder, wird der Event-Handler `onFocus=` des jeweiligen Feldes aktiv. Dabei wird in das jeweilige Feld eine Aufforderung geschrieben, was in dem Feld einzugeben ist.

### JS 1.2 4.0 4.x **onKeyDown (bei gedrückter Taste)**

Tritt ein, wenn der Anwender, während er ein Element aktiviert hat, eine Taste drückt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<input>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
function Aktualisieren() {
    document.Test.Kontrolle.value = document.Test.Eingabe.value.length + 1;
    return true;
}
</script>
</head><body>
<form name="Test" action="">
Kurzbeschreibung Ihrer Homepage (max 50 Zeichen):<br>
<input type="text" name="Eingabe" size="40" onKeyDown="Aktualisieren(this.value)">
<input type="text" value="0" readonly size="3" name="Kontrolle"><br>
<input type="reset">
</form>
</body></html>
```



18.12: Das rechte Eingabefeld zeigt sofort an, wie viele Zeichen links eingetippt wurden.

Im Beispiel wird ein Formular definiert, in dem der Anwender eine Kurzbeschreibung seiner Homepage in einem Eingabefeld abliefern kann. Der Text soll maximal 50 Zeichen lang sein. Damit der Anwender nicht mitzählen muss, gibt es ein kleines Eingabefeld nebendran, in dem nach jedem Tastendruck ausgegeben wird, wie viele Zeichen bereits eingegeben wurden. Dazu ist in dem Eingabefeld mit Namen `Eingabe` der Event-Handler `onKeyDown=` notiert. Er bewirkt, dass, solange der Anwender in dem Formularfeld etwas eingibt, bei jedem Tastendruck die Funktion `Aktualisieren()` aufgerufen wird, die im Dateikopf in einem Script-Bereich definiert ist. Diese Funktion errechnet aus `document.Test.Eingabe.value.length + 1`, wie viele Zeichen bereits eingegeben wurden, und schreibt eine entsprechende Ausgabe in das dafür vorgesehene »Eingabe«-Feld.


**onKeyPress (bei gedrückt gehaltener Taste)**

Tritt ein, wenn der Anwender eine Taste drückt und diese gedrückt hält. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<input>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input type="text" size="30" name="Eingabe"
  onKeyPress="alert(this.value)">
</form>
</body></html>
```

Im Beispiel ist ein Formular mit einem Eingabefeld definiert. Wenn der Anwender in dem Eingabefeld etwas eingibt, wird bei jedem Tastendruck in einem Meldungsfenster der aktuelle Wert des Formularfeldes ausgegeben. Dazu ist in dem Eingabefeld der Event-Handler `onKeyPress` notiert. Er tritt in Aktion, wenn eine Taste gedrückt und gedrückt gehalten wird.

In Netscape 4.x und Opera 5 wird dieser Event-Handler stets **nach** dem Event-Handler `onKeyDown` ausgeführt. Gibt der Event-Handler `onKeyDown` den Wert `false` zurück, so wird unter Netscape die Ausführung des Event-Handlers `onKeyPress` unterdrückt. Im MS Internet Explorer unterdrückt der Event-Handler `onKeyPress` den Event-Handler `onKeyDown`.

**Beachten Sie:** Im MS Internet Explorer können Sie zusätzlich die Eigenschaften des event-Objekts abfragen.


**onKeyUp (bei losgelassener Taste)**

Tritt ein, wenn der Anwender eine Taste gedrückt hat und diese wieder loslässt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<input>` `<textarea>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input type="text" size="30" name="Eingabe"
  onKeyUp="this.form.Ausgabe.value=this.value"><br>
```

```
<input type="text" readonly size="30" name="Ausgabe"><br>
<input type="reset">
</form>
</body></html>
```

Im Beispiel wird ein Formular definiert, das zwei Eingabefelder mit den Namen `Eingabe` und `Ausgabe` enthält. Wenn der Anwender in dem oberen Feld, also dem, das für die Eingabe gedacht ist, etwas eingibt, wird der Wert automatisch Zeichen für Zeichen in das untere, also das Ausgabefeld übernommen. Dazu ist im Eingabefeld der Event-Handler `onKeyUp` notiert. Dieser Event-Handler tritt in Aktion, wenn der Anwender in dem Feld eine Taste gedrückt und wieder losgelassen hat, was ja bei jedem eingegebenen Zeichen der Fall ist. Mit `this.form.Ausgabe.value=this.value` wird dem Ausgabefeld der aktuelle Wert des Eingabefeldes zugewiesen.



### **onLoad (beim Laden einer Datei)**

Tritt ein, wenn eine HTML-Datei geladen wird. Nach JavaScript (Netscape) und HTML 4.0 erlaubt in folgenden HTML-Tags: `<frameset>` `<body>`. Ein Beispiel:

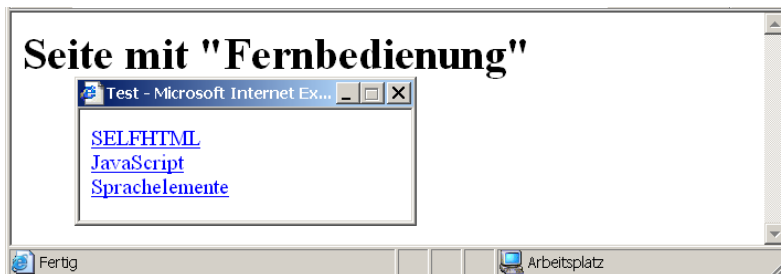
```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function NaviFenster() {
  Navigation = window.open("navigat.htm","Navigation","height=100,width=300");
  Navigation.focus();
}
// -->
</script>
</head>
<body onLoad="NaviFenster()">
<h1>Seite mit "Fernbedienung"</h1>
</body></html>
```

Beispiel – Datei *navigat.htm*:

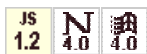
```
<html><head><title>Test</title>
</head><body>
<a href= "javascript:void(opener.location.href='../ ../index.htm');">
SELFHTML</a><br>
<a href= "javascript:void(opener.location.href='../ ../index.htm');">
JavaScript</a><br>
<a href= "javascript:void(opener.location.href='../index.htm');">
Sprachelemente</a><br>
</body></html>
```

Im Beispiel wird beim Einlesen der HTML-Datei ein zweites Fenster geöffnet, das zum Beispiel als »Fernbedienung« des Hauptfensters dienen könnte. Dazu ist im einleitenden `<body>`-Tag der Event-Handler `onLoad` notiert. Er ruft die Funktion `NaviFenster()` auf, die in einem JavaScript-Bereich im Dateikopf definiert ist. Innerhalb dieser Funktion steht der Befehl zum Öffnen des Zweitfensters. Das Fenster erhält auch gleich den Fokus (wird zum aktiven Fenster), sodass

es im Vordergrund des Hauptfensters zu sehen ist. Ins Zweitfenster wird die Datei `navigat.htm` geladen. Diese enthält Verweise mit dem »Event-Handler« `javascript:.`. Beim Ausführen des Verweises wird im Elternfenster des Zweitfensters, markiert durch den reservierten Fenstername `opener`, mit `document.location.href` eine Seite geladen. Die Anweisung `opener.location.href='../index.htm'` gibt den URI der zu ladenden Datei zurück. Mit Hilfe des Operators `void` wird dieser unterdrückt. Dadurch werden Anzeigefehler vermieden.



18.13: Beim Laden des Beispiels öffnet sich ein zusätzliches Fenster mit Navigationslinks.



### ***onMousedown (bei gedrückter Maustaste)***

Tritt ein, wenn der Anwender die Maustaste gedrückt hält. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<input type="button">` `<a>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="../../../index.htm"
onMousedown="window.status='Verweis zur Startseite';return true;">Verweis</a>
</body></html>
```

Das Beispiel enthält einen Verweis. Bei vollständig erfolgreichem Anklicken des Verweises wird ganz normal das Verweisziel, im Beispiel also `../../../index.htm`, aufgerufen. Vorher jedoch, sobald der Anwender die Maustaste gedrückt und bevor er sie wieder losgelassen hat, tritt der Event-Handler `onMousedown=` in Aktion, der im Verweis-Tag notiert ist. Im Beispiel wird dabei in der Statuszeile des Browsers ausgegeben: Verweis zur Homepage.



### ***onMousemove (bei weiterbewegter Maus)***

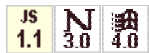
Tritt ein, wenn der Anwender die Maus bewegt, unabhängig davon, ob die Maustaste gedrückt ist oder nicht. Nach JavaScript (Netscape) bislang nicht in HTML-Tags möglich, sondern nur im Zusammenhang mit dem `event`-Objekt. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>`

<caption> <center> <cite> <code> <col> <colgroup> <dd> <del> <dfn> <dir> <div>  
 <dl> <dt> <em> <fieldset> <form> <h1> <h2> <h3> <h4> <h5> <h6> <hr> <i> <img>  
 <input> <ins> <kbd> <label> <legend> <li> <link> <map> <menu> <noframes>  
 <noscript> <object> <ol> <optgroup> <option> <p> <pre> <q> <s> <samp> <select>  
 <small> <span> <strike> <strong> <sub> <sup> <table> <tbody> <td> <textarea>  
 <tfoot> <th> <thead> <tr> <tt> <u> <ul> <var>. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript">
function Mauskontrolle(Element) {
  var Pos = Element.offsetLeft + "/" + Element.offsetTop;
  window.status = Pos;
  return true;
}
</script>
</head><body>
<p onmousemove="Mauskontrolle(this)">Hier ein kleiner Text</p>
</body></html>
```

Das Beispiel zeigt, wie Event-Handler auch in HTML-Tags funktionieren, bei denen das früher nicht möglich war. In dem Beispiel wird ein Textabsatz definiert. Innerhalb des Textabsatzes ist der Event-Handler `onmousemove=` notiert. Der Event-Handler tritt in Aktion, solange der Anwender die Maus im Anzeigebereich des Textabsatzes bewegt. Dann wird so oft wie möglich die Funktion `Mauskontrolle()` aufgerufen, die in einem Scriptbereich im Dateikopf notiert ist. Dieser Funktion wird als Parameter mittels `this` das betroffene Element übergeben. Die Funktion bewirkt, dass in der Statuszeile des Browsers jeweils die Koordinaten der linken oberen Ecke des Elementes angezeigt werden.

**Beachten Sie:** Im MS Internet Explorer können Sie zusätzlich die Eigenschaften des event-Objekts abfragen.



#### ***onMouseout (beim Verlassen des Elements mit der Maus)***

Tritt ein, wenn der Anwender mit der Maus über ein Element fährt und dieses dabei verlässt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: <a> <area>. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: <a> <abbr> <acronym> <address> <area> <b> <big> <blockquote> <body> <button> <caption> <center> <cite> <code> <col> <colgroup> <dd> <del> <dfn> <dir> <div> <dl> <dt> <em> <fieldset> <form> <h1> <h2> <h3> <h4> <h5> <h6> <hr> <i> <img> <input> <ins> <kbd> <label> <legend> <li> <link> <map> <menu> <noframes> <noscript> <object> <ol> <optgroup> <option> <p> <pre> <q> <s> <samp> <select> <small> <span> <strike> <strong> <sub> <sup> <table> <tbody> <td> <textarea> <tfoot> <th> <thead> <tr> <tt> <u> <ul> <var>. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="http://selfaktuell.teamone.de/news.htm"
onmouseout="alert('Die News sollten Sie ruhig mal besuchen')"><b>News</b></a>
</body></html>
```



Im Beispiel ist ein Verweis definiert. Wenn der Anwender mit der Maus über den Verweis fährt, tritt der Event-Handler `onMouseover=` in Aktion. Im Beispiel wird dann ein Meldungsfenster ausgegeben, das den Anwender darauf hinweist, dass er die News-Seite ruhig mal aufrufen soll.

### 18.10.1 `onMouseover` (beim Überfahren des Elements mit der Maus)

Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<a>` `<area>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>` `<map>` `<menu>` `<noframes>` `<noscript>` `<object>` `<ol>` `<optgroup>` `<option>` `<p>` `<pre>` `<q>` `<s>` `<samp>` `<select>` `<small>` `<span>` `<strike>` `<strong>` `<sub>` `<sup>` `<table>` `<tbody>` `<td>` `<textarea>` `<tfoot>` `<th>` `<thead>` `<tr>` `<tt>` `<u>` `<ul>` `<var>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<h1 id="Test"
  onMouseover="this.innerHTML='Sehen Sie?'"
  onMouseout="this.innerHTML='Ich bin dynamisch'">Ich bin dynamisch</h1>
</body></html>
```

Das Beispiel zeigt, wie Event-Handler auch in HTML-Tags funktionieren, bei denen das bislang nicht möglich war. Das Beispiel funktioniert mit dem MS Internet Explorer ab Version 4.x und im Netscape Navigator ab Version 6, welche die Event-Handler nach HTML 4.0 weitgehend interpretieren. In dem Beispiel wird eine Überschrift erster Ordnung definiert. Innerhalb der Überschrift sind die Event-Handler `onMouseover=` und `onMouseout=` notiert. Der Event-Handler `onMouseover=` tritt in Aktion, wenn der Anwender die Maus in den Anzeigebereich der Überschrift bewegt, und `onMouseout=` wird aktiv, wenn er die Maus wieder aus dem Anzeigebereich herausbewegt. Mit Hilfe von `this` nehmen Sie Bezug auf das *objekte.htm* – *this* und können mittels der Eigenschaft `innerHTML` mit jedem Aktivwerden eines der beiden Event-Handler den Text der Überschrift dynamisch austauschen. Bei `onMouseover=` wird ein anderer Text angezeigt, bei `onMouseout=` wieder der ursprüngliche Text.

**Beachten Sie:** Auch Opera ab Version 5 interpretiert den Event-Handler nach HTML 4.0. Er kennt jedoch nicht die Eigenschaft `innerHTML`.

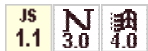
### `onMouseUp` (bei losgelassener Maustaste)

Tritt ein, wenn der Anwender die Maustaste gedrückt hat und sie nun wieder loslässt. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<input type="button">` `<a>`. Nach HTML 4.0 erlaubt in folgenden HTML-Tags: `<a>` `<abbr>` `<acronym>` `<address>` `<area>` `<b>` `<big>` `<blockquote>` `<body>` `<button>` `<caption>` `<center>` `<cite>` `<code>` `<col>` `<colgroup>` `<dd>` `<del>` `<dfn>` `<dir>` `<div>` `<dl>` `<dt>` `<em>` `<fieldset>` `<form>` `<h1>` `<h2>` `<h3>` `<h4>` `<h5>` `<h6>` `<hr>` `<i>` `<img>` `<input>` `<ins>` `<kbd>` `<label>` `<legend>` `<li>` `<link>`

<map> <menu> <noframes> <noscript> <object> <ol> <optgroup> <option> <p> <pre> <q>  
<s> <samp> <select> <small> <span> <strike> <strong> <sub> <sup> <table> <tbody>  
<td> <textarea> <tfoot> <th> <thead> <tr> <tt> <u> <ul> <var>. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="../../index.htm"
  onMouseup="alert('Die Show beginnt JETZT');
window.location.href='../../index.htm'; return false">Verweis</a>
</body></html>
```

Im Beispiel ist ein Verweis notiert. Wenn der Anwender den Verweis anklickt und die Maustaste loslässt, also unmittelbar vor dem Laden der Seite, auf die verwiesen wird, geht ein Meldungsfenster auf, das den Anwender in aller Dramatik noch mal darauf hinweist, dass jetzt die Show beginnt. Dazu ist im einleitenden Verweis-Tag der Event-Handler `onMouseup=` notiert. Wenn der in Aktion tritt, wird zunächst das Meldungsfenster angezeigt. Anschließend wird die Datei `../../index.htm` geladen. Die Anweisung `return false` verhindert, dass Browser, die den Event-Handler `onMouseup=` interpretieren, den `location.href`-Verweis **nicht** ausführen, sondern den gewöhnlichen Verweis, der beim `href`-Attribut notiert ist.



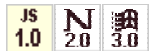
#### ***onReset (beim Zurücksetzen des Formulars)***

Tritt ein, wenn der Anwender Eingaben in einem Formular verwerfen will. Nach JavaScript (Netscape) erlaubt in folgendem HTML-Tag: `<form>`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
function ResetCheck() {
  var chk = window.confirm("Wollen Sie alle Eingaben loeschen?");
  return(chk);
}
</script>
</head><body>
<form name="Test" onReset="return ResetCheck()" action=""><pre>
Name: <input size="30"><br>
Idee: <input size="30"><br>
<input type="reset">
</pre></form>
</body></html>
```

Das Beispiel enthält ein Formular, das unter anderem einen Abbrechen-Button (Reset-Button) enthält. Beim Anklicken dieses Buttons werden normalerweise alle Eingaben im Formular gelöscht. Im Beispiel ist jedoch im einleitenden `<form>`-Tag der Event-Handler `onReset=` notiert. Dieser tritt in Aktion, wenn der Reset-Button angeklickt wird. Im Beispiel wird dann die Funktion `ResetCheck()` aufgerufen, die in einem Script-Bereich im Dateikopf steht. Diese Funktion fragt den Anwender in einem Bestätigungsfenster (`window.confirm()`), ob er wirklich alle Eingaben in dem Formular löschen will. Bestätigt er den Löschwunsch, gibt das Bestätigungsfenster den Wert `true` zurück. Verneint er den Löschwunsch, wird `false` zurückgegeben. Der Rückgabewert wird in der Variablen `chk` gespeichert und diese wird wiederum von der Funktion

ResetCheck() an den aufrufenden Event-Handler zurückgegeben. Der Effekt ist, dass die Formulareingaben nur gelöscht werden, wenn true zurückgegeben wird.



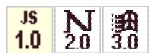
### **onSelect (beim Selektieren von Text)**

Tritt ein, wenn der Anwender Text selektiert. Nach JavaScript (Netscape) und HTML 4.0 erlaubt in folgenden HTML-Tags: <input> <textarea>. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<textarea cols="40" rows="10" name="Eingabe" onSelect="window.status='Mit der
rechten Maustaste über dem selektierten Text klicken!'; return true;">Selektieren
Sie einfach irgendwelchen Text aus diesem Text hier!</textarea>
</form>
</body></html>
```

Im Beispiel wird ein Formular mit einem mehrzeiligen Eingabefeld definiert, das mit Text vorgelegt ist. Wenn der Anwender Text in diesem Feld selektiert, wird mit onSelect erreicht, dass in der Statuszeile (window.status) ein Hinweis ausgegeben wird, was der Anwender nun tun kann.

**Beachten Sie:** Opera 5 und Netscape 4.x interpretieren diesen Event-Handler nicht.

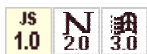


### **onSubmit (beim Absenden des Formulars)**

Tritt ein, wenn der Anwender ein Formular absendet. Nach JavaScript (Netscape) erlaubt in folgendem HTML-Tag: <form>. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CheckInput() {
for(i=0; i<document.forms[0].elements.length; ++i)
if(document.forms[0].elements[i].value == "") {
alert("Es wurden nicht alle Felder ausgefüllt!");
document.forms[0].elements[i].focus();
return false;
}
return true;
}
//-->
</script>
</head><body>
<form action="onsubmit.htm" onSubmit="return CheckInput();">
Feld 1: <input size="30"><br>
Feld 2: <input size="30"><br>
Feld 3: <input size="30"><br>
<input type="submit">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit mehreren Eingabefeldern. Beim Absenden des Formulars, also beim Klicken auf den Submit-Button, wird jedoch erst mal überprüft, ob in allen Feldern etwas eingegeben wurde. Wenn eines der Felder leer gelassen wurde, wird das Formular nicht abgesendet. Stattdessen wird eine Fehlermeldung ausgegeben, und der Cursor wird in das erste nicht ausgefüllte Eingabefeld positioniert. Dazu ist im einleitenden `<form>`-Tag der Event-Handler `onSubmit=` notiert. Beim Absenden des Formulars wird dadurch die Funktion `CheckInput()` aufgerufen, die in einem Script-Bereich im Dateikopf notiert ist. Diese Funktion prüft in einer `for`-Schleife alle einzelnen Formularfelder darauf, ob sie einen leeren Inhalt haben (leere Zeichenkette ""). Ist das der Fall, wird die Fehlermeldung ausgegeben und auf das entsprechende Formularfeld positioniert. An den aufrufenden Event-Handler `onSubmit` wird der Wert `false` zurückgegeben. Nur wenn alle Formularfelder ausgefüllt wurden, wird `true` zurückgegeben. Dadurch wird entschieden, ob das Formular abgeschickt wird oder nicht.



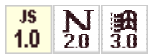
### *onUnload (beim Verlassen der Datei)*

Tritt ein, wenn eine HTML-Datei verlassen wird. Nach JavaScript (Netscape) und HTML 4.0 erlaubt in folgenden HTML-Tags: `<frameset>` `<body>`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
var Start = new Date();
var Startzeit = Start.getTime();

function Aufenthalt() {
  var Ende = new Date();
  var Endzeit = Ende.getTime();
  var Aufenthalt = Math.floor((Endzeit - Startzeit) / 1000);
  alert("Sie waren " + Aufenthalt + " Sekunden auf dieser Seite");
}
</script>
</head>
<body onUnload="Aufenthalt()">
<h1>Einen langen Aufenthalt auf dieser Seite!</h1>
<p><a href="../../index.htm"><b>Und dann aber nix wie weg!</b></a></p>
</body></html>
```

Im Beispiel ist im Dateikopf ein JavaScript-Bereich definiert. Gleich beim Einlesen der Datei wird mit Hilfe des `Date`-Objekts der aktuelle Zeitpunkt ermittelt und in der Variablen `Startzeit` gespeichert. Im einleitenden `<body>`-Tag der Datei ist der Event-Handler `onUnload=` notiert. Er tritt in Aktion, wenn die Datei – zum Beispiel durch Anklicken eines Verweises zu einer anderen Seite – verlassen wird. In diesem Fall wird im Beispiel die Funktion `Aufenthalt()` aufgerufen, die ebenfalls in dem Script-Bereich im Dateikopf steht. Diese Funktion ermittelt wieder den aktuellen Zeitpunkt, ermittelt dann aber noch die Differenz zwischen gespeicherter Startzeit und der jetzt ermittelten »Endzeit« und gibt das Ergebnis in einem Meldungsfenster aus.

**javascript: (bei Verweisen)**

Dies ist kein Event-Handler im engeren Sinn. Es handelt sich um eine Syntax, die eingeführt wurde, um JavaScript-Code als Verweisziel zu notieren. Nach JavaScript (Netscape) erlaubt in folgenden HTML-Tags: `<a>` `<area>`. Ein Beispiel:

```
<html><head><title>Test</title>
</head>
<body>
<a href="javascript:alert(document.lastModified)"><b>Letzter Update</b></a>
</body></html>
```

Das Beispiel gibt bei Anklicken des Verweises ein Meldungsfenster mit dem Zeitpunkt des letzten Updates der Datei aus. Um einen Verweis dieser Art zu notieren, notieren Sie hinter dem Attribut `href=` in Anführungszeichen das Schlüsselwort `javascript`, gefolgt von einem Doppelpunkt `:` und dahinter eine oder mehrere JavaScript-Anweisungen. Bei mehreren Anweisungen ist es jedoch besser, diese in einer Funktion zu notieren und beim Verweis dann diese Funktion aufzurufen.

**Beachten Sie:** Diese Form des Aufrufes erfordert zwingend den Rückgabewert `undefined`. Ist dies nicht der Fall, wird der jeweilige Rückwert der JavaScript-Anweisung in das Dokument geschrieben und die aktuelle Seite gelöscht. Sie können gegebenenfalls unter Verwendung des Operators `void` den Rückgabewert einer Anweisung unterdrücken und auf `undefined` setzen.

## 19 JavaScript: Beispiele

Am besten lernt man an Hand klarer, praktischer Beispiele. Auf den folgenden Seiten finden Sie eine Reihe von ausführlichen Beispielen:

- Zwei Frames gleichzeitig ändern
- Seitenanzeige in Frames verhindern
- Taschenrechner
- Monatskalender
- Persönliche Seitenbesuche zählen mit Cookies
- Formulareingaben überprüfen
- Dynamische grafische Buttons
- Aufenthaltsdauer auf Web-Seite anzeigen
- Verweisliste als Auswahlliste
- Fehlerbehandlung mit dem Event-Handler `onError`
- Fehlerbehandlung mit `try..catch`
- Änderungen der Fenstergröße überwachen

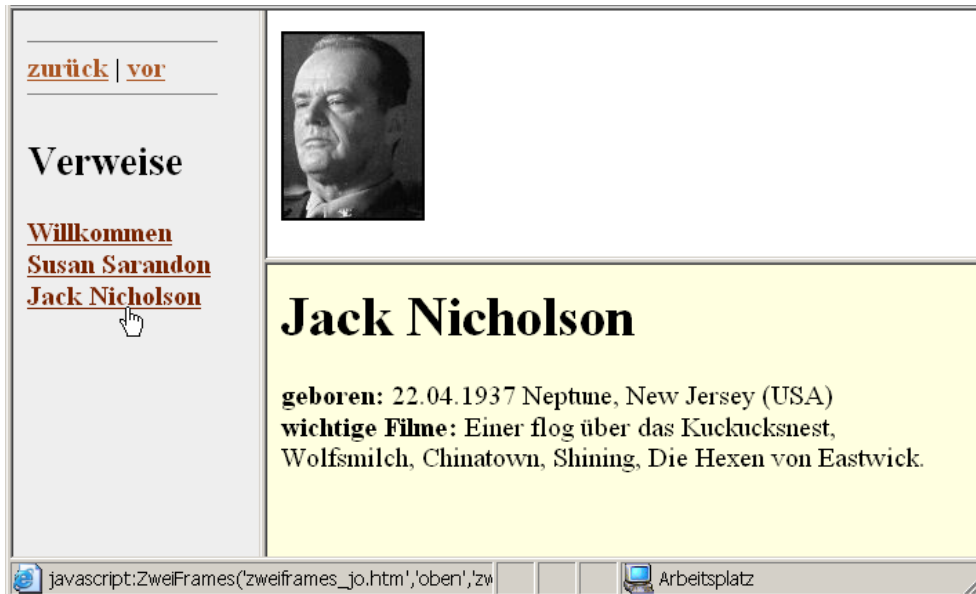
Weitere Beispiele zu JavaScript enthält das Kapitel 21 über Dynamisches HTML.

### 19.1 Zwei Frames gleichzeitig ändern

Wer schon einmal ein HTML-Projekt mit mehreren Frames erstellt hat, kann nachvollziehen, dass es manchmal wünschenswert ist, beim Ausführen eines Verweises nicht nur den Inhalt eines, sondern zweier oder mehrerer Frame-Fenster gleichzeitig zu ändern. Nun kann man in solchen Fällen zwar auf eine Datei verweisen, die ein neues, entsprechendes Frameset definiert. Aber wenn Sie viele solcher Verweise haben, wird dieses Verfahren schnell aufwändig. Als Alternative dazu können Sie sich mit einem JavaScript behelfen.

Ein Nachteil von »zwei Frames gleichzeitig ändern« mit JavaScript ist allerdings, dass dabei die History-Funktion im Browser nicht mehr so arbeitet wie gewünscht. Denn aus Sicht des Browsers wurden zwei Verweise ausgeführt, doch ein History-Eintrag springt nur eine Seite zurück. Der Anwender muss also zweimal auf den Zurück-Button des Browsers klicken, um wieder dahin zu kommen, wo er vor dem Anklicken des Verweises war. Eine Möglichkeit, dieses unerwünschte Verhalten auszuschalten, ist, selber Verweise anzubieten, die das »Back« und »Forward« im Browser auslösen, jedoch so, dass sich dabei wie gewünscht ebenfalls zwei Frames

gleichzeitig ändern. Das Beispiel auf dieser Seite zeigt deshalb auch, wie sich so etwas realisieren lässt.



19.1: Klicken Sie links einen Link an, so ändern sich beide Frame-Inhalte auf der rechten Seite.



### Quelltext mit Erläuterungen

Das Beispiel zeigt die Datei mit den Frameset-Definitionen und die Datei mit den Verweisen. Die übrigen Dateien, die im Anzeigebeispiel aufgerufen werden, sind für das Verständnis ohne Bedeutung.

Beispiel – Datei *zweiframes.htm*:

```
<html>
<head>
<title>Zwei Frames gleichzeitig &auuml;ndern</title>
</head>
<frameset cols="180,*">
  <frame src="zweiframes_links.htm" name="links">
  <frameset rows="180,*">
    <frame src="zweiframes_to.htm" name="oben">
    <frame src="zweiframes_tu.htm" name="unten">
  </frameset>
</frameset>
</html>
```

Beispiel – Datei *zweiframes\_links.htm*:

```
<html>
<head>
<title>ZweiFrames - Verweise</title>
<script type="text/javascript">
<!--
function ZweiFrames(URI1,F1,URI2,F2) {
    Frame1=eval("parent."+F1);
    Frame2=eval("parent."+F2);
    Frame1.location.href = URI1;
    Frame2.location.href = URI2;
}
function ZweiFramesBack(F1,F2) {
    Frame1=eval("parent."+F1);
    Frame2=eval("parent."+F2);
    Frame1.history.back();
    if (!window.opera) window.setTimeout("Frame2.history.back()",10);
}
function ZweiFramesForward(F1,F2) {
    Frame1=eval("parent."+F1);
    Frame2=eval("parent."+F2);
    Frame1.history.forward();
    if (!window.opera) window.setTimeout("Frame2.history.forward()",10);
}
//-->
</script>
</head>
<body bgcolor="#EEEEEE" text="#000000" link="#AA5522" vlink="#772200"
alink="#000000">

<div>
<hr noshade size="1">
<a href="javascript:ZweiFramesBack('oben','unten')"><b>zur&uuml;ck</b></a>
<b>|</b>
<a href="javascript:ZweiFramesForward('oben','unten')"><b>vor</b></a>
<hr noshade size="1">
</div>

<h2>Verweise</h2>
<p>
<a
href="javascript:ZweiFrames('zweiframes_to.htm','oben','zweiframes_tu.htm','unten')">
<b>Willkommen</b>
</a>
<br>
<a
href="javascript:ZweiFrames('zweiframes_so.htm','oben','zweiframes_su.htm','unten')">
```



```

<b>Susan Sarandon</b>
</a>
<br>
<a
href="javascript:ZweiFrames('zweiframes_jo.htm','oben','zweiframes_ju.htm','unten')">
<b>Jack Nicholson</b>
</a>
</p>

</body>
</html>

```

Der erste Teil des Beispiels zeigt die Datei *zweiframes.htm*, die das Frameset für eine Schauspielerinfothek bereitstellt. Das Frameset ist dreigeteilt: links ein Frame-Fenster mit Namen *links* für Verweise, und rechts zwei Frames mit den Namen *oben* und *unten* für Informationen.

Der zweite Teil des Beispiels zeigt die Datei *zweiframes\_links.htm*, die ja, wie in der Datei *zweiframes.htm* ersichtlich, gleich zu Beginn in das linke Frame-Fenster geladen wird. Diese Datei enthält die Verweise. Alle Verweise ändern den Inhalt von zwei Frame-Fenstern gleichzeitig, nämlich jeweils den der beiden rechten Frames mit den Namen *oben* und *unten*. Deshalb ist in dieser Datei auch der JavaScript-Code für das Ändern der beiden anderen Frames notiert.

Um die Funktionsweise zu verstehen, ist es sinnvoll, zunächst einmal die HTML-Verweise im *<body>*-Bereich der Datei *zweiframes\_links.htm* zu betrachten. Alle Verweise enthalten beim *href*-Attribut JavaScript-Code, eingeleitet durch den Pseudo-Event-Handler *javascript:.* Dahinter folgt jeweils der Aufruf einer Funktion. Bei dem Verweis mit dem Verweistext *zurück* wird die Funktion *ZweiFramesBack()* aufgerufen, bei dem Verweis mit dem Verweistext *vor* die Funktion *ZweiFramesForward()*, und bei den übrigen Verweisen die Funktion *ZweiFrames()*. In allen Fällen werden der jeweils aufgerufenen Funktion die Namen zweier vorhandener Frames als Parameter übergeben, nämlich *'oben'* und *'unten'*.

Im Kopf der HTML-Datei, also zwischen *<head>* und *</head>*, wird mit *<script type="text/javascript"> .... </script>* ein JavaScript-Bereich definiert. Innerhalb dieses Bereichs sind die drei Funktionen *ZweiFrames()*, *ZweiFramesBack()* und *ZweiFramesForward()* notiert, die von den Verweisen in der Datei aufgerufen werden.

Die Funktion *ZweiFrames()* erwartet vier Parameter:

1. den URI des gewünschten Inhalts des ersten Frame-Fensters (*URI1*),
2. den Namen des gewünschten ersten Frame-Fensters innerhalb des Framesets (*F1*),
3. den URI des gewünschten Inhalts des zweiten Frame-Fensters (*URI2*),
4. den Namen des gewünschten zweiten Frame-Fensters innerhalb des Framesets (*F2*).

Dadurch ist die Funktion allgemein gehalten, d.h. Sie können diese Funktion auch in ganz anderen Framesets zum gleichen Zweck verwenden. Innerhalb der Funktion stehen zunächst zwei Aufrufe der Funktion *eval*. Dies ist nötig, damit der JavaScript-Interpreter die übergebenen Namen der Frame-Fenster nicht einfach als beliebige Zeichenkette interpretiert, sondern versucht, als vorhandene Objekte zu erkennen. Die Aufrufe sind so gestaltet, dass in den Variablen

Frame1 bzw. Frame2 im Beispiel hinterher die Objekte `parent.oben` bzw. `parent.unten` gespeichert sind. Solchermaßen ausgerüstet, lassen sich die beiden Variablen anschließend in den Aufrufen `Frame1.location.href` und `Frame2.location.href` wie die entsprechenden Objekte verwenden, sodass im Beispiel eigentlich ausgeführt wird: `parent.oben.location.href` bzw. `parent.unten.location.href`. Die Eigenschaft `location.href` bewirkt einen JavaScript-Verweis zu einer anderen Adresse. In der Funktion werden den beiden Aufrufen dieser Eigenschaft einfach die beiden beim Funktionsaufruf übergebenen Adressen `URI1` bzw. `URI2` zugewiesen.

Die Funktionen `ZweiFramesBack()` und `ZweiFramesForward()` sind ähnlich aufgebaut. Allerdings benötigen sie nur zwei Parameter, nämlich die Namen der betroffenen Frame-Fenster (`F1` und `F2`). Mit Aufrufen der Objektmethoden `history.back()` bzw. `history.forward()` erreichen Sie die Funktionalität der entsprechenden Buttons »Zurück« und »Vor« im Browser. Dass der jeweils zweite Aufruf der Objektmethoden noch mal in ein `setTimeout()` mit einer geringen Verzögerung von 1/100 Sekunde gesetzt ist, hat pragmatische Gründe: manche Versionen des Internet Explorers würden sonst »vergessen«, den zweiten Aufruf durchzuführen.

**Beachten Sie:** Opera 5.12 interpretiert die Methode `history.back()` auf das `_top`-Fenster bezogen und ignoriert die Angabe des Zielfensters beim Aufruf der Methode. Das gilt auch, wenn Sie die Methode innerhalb eines Frames selbst aufrufen. Der Browser hält sich strikt an die Reihenfolge der History-Einträge im Hauptfenster. Wurden mittels JavaScript zwei Frames gleichzeitig geändert, betrachtet Opera diese Änderung als einen Eintrag und geht in der History zwei Schritte zurück. Aus diesem Grund gilt es zu verhindern, dass Opera ein zweites Mal ein `history.back()` ausführt. Dies wird im obigen Beispiel mit `if (!window.opera)` erreicht. Nur ein Opera-Browser kennt das Objekt `window.opera`. In der bedingten Anweisung wird gefragt, ob der Browser dieses Objekt **nicht** kennt. Nur in diesem Fall wird die Objektmethode ein zweites Mal aufgerufen.

## 19.2 Seitenanzeige in Frames verhindern

Wenn Sie eigene WWW-Seiten erstellen, werden Sie kaum wollen, dass Ihr prachtvolles Werk in einem kleinen Fenster einer fremden Seite angezeigt wird. Dies ist jedoch mit Hilfe von Frames problemlos möglich, und es gibt einige Web-Guide-Projekte, die unverschämt genug sind, fremde Projekte als »Schaufenster« im eigenen Projekt anzuzeigen. Lesen Sie dazu auch den Abschnitt 9.5.2 über unfaire Schaufenster-Effekte.

Mit einem kleinen JavaScript können Sie erzwingen, dass Ihre eigenen Seiten beim Aufruf in jedem Fall das gesamte Browser-Fenster ausfüllen. Das Script erzwingt die Anzeige einer HTML-Datei im Vollbild, wenn eine andere Seite versucht, die Datei in ein Frame-Fenster zu laden. Auf »normale« Verweise und Aufrufe der Datei hat das Script keine Auswirkungen.



### Quelltext mit Erläuterungen

Das Beispiel zeigt die Datei, die das Geladenwerden in ein Frameset verhindert. Die Dateien, die im Anzeigebeispiel zum Frameset gehören, sind für das Verständnis ohne Bedeutung.

```

<html>
<head>
<title>Seitenanzeige in Frames verhindern</title>
<script type="text/javascript">
<!--
  if(top!=self)
    top.location=self.location;
//-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000" link="#AA5522" vlink="#772200"
alink="#000000">

<h1>Keine Chance für Schaufenster-Anbieter!</h1>

</body>
</html>

```



19.2: Wird die Beispieldatei in das Frame rechts geladen, so übernimmt sie die Kontrolle und füllt das ganze Browser-Fenster.

Das kleine JavaScript im Beispiel sollte im Kopf jeder HTML-Datei stehen, die Sie nicht in fremden Frames sehen möchten. Das Script wird beim Laden der Datei automatisch ausgeführt, da der Code nicht in einer Funktion gebunden ist. In dem Code – er besteht aus einer einzigen bedingten Anweisung – wird mit `top!=self` abgefragt, ob die oberste Referenzebene der eigenen Ebene entspricht. Ist `self`, also das eigene Fenster, gleichzeitig auch das oberste Fenster (`top`), dann ist alles in Ordnung, und die Anweisung, die von der Bedingung abhängt, wird nicht ausgeführt. Ist dagegen `self` **nicht** das gleiche wie `top`, bedeutet dies, dass irgendwelche Frames angezeigt werden. In diesem Fall wird die Anweisung `top.location=self.location` ausgeführt. Mit dieser Anweisung erzwingen Sie, dass die aktuelle Datei auf jeden Fall im gesamten Anzeigefenster des Browsers angezeigt wird.

**Beachten Sie:** Bei Opera 5.12 haben Sie keinen Zugriff auf die oberste Fensterebene `top`, wenn diese nicht zur eigenen Domain gehört. Sie können jedoch über einen Umweg auch diesen Browser dazu veranlassen, das Frameset zu sprengen. Dazu müssen Sie dafür sorgen, dass ein Verweis mit der Eigenschaft `target="_top"` angeklickt wird. Ein Beispiel:

```
<html>
<head>
<title>Seitenanzeige in Frames verhindern</title>
<script type="text/javascript">
<!--

    if(top!=self)
    {
        if (window.opera) window.onload=opera_befreien;
        else top.location=self.location;
    }

function opera_befreien()
{
    document.links['befreien'].click();
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000" link="#AA5522" vlink="#772200"
alink="#000000">

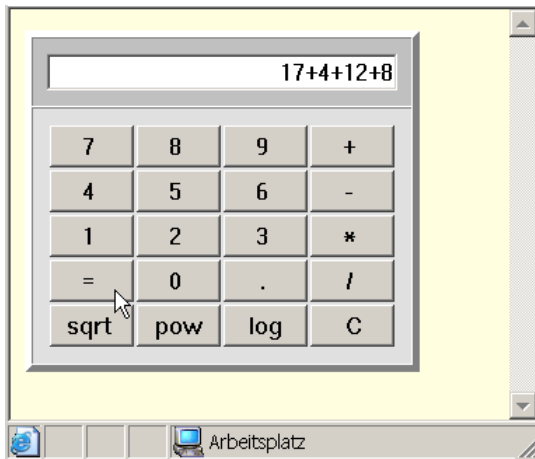
<a name="befreien" href="seitenanzeige_opera.htm" target="_top"></a>

<h1>Keine Chance für Schaufenster-Anbieter!</h1>
</body>
</html>
```

Innerhalb der Seite befindet sich ein unsichtbarer Verweis (Verweis ohne Verweistext) namens `befreien`. Dieser Verweis besitzt als Zielfenster `_top` und als Verweisziel die Seite selbst. Wird nun beim Laden der Seite festgestellt, dass diese sich innerhalb eines fremden Framesets befindet, so verwenden alle Browser außer Opera das oben beschriebene Script. In Opera dagegen wird lediglich der Event-Handler `onload` initialisiert. Der Aufruf `window.onload=opera_befreien` hat die Aufgabe, nach dem Laden der Seite die Funktion `opera_befreien()` aufzurufen. Innerhalb der Funktion passiert nichts weiter, als dass der Verweis mit der Methode `click()` angeklickt wird. Diese undokumentierte Methode funktioniert mit allen Identifikationseinstellungen von Opera. Das Verweisziel lädt sich in die oberste Fensterebene und sprengt das Frameset.

### 19.3 Taschenrechner

Das Beispiel baut mit Hilfe eines HTML-Formulars einen halbwegs »echt« aussehenden Taschenrechner auf. Der Anwender kann diesen Taschenrechner wie üblich bedienen. Allerdings bietet dieser Taschenrechner im Gegensatz zu normalen Taschenrechnern auch die Möglichkeit an, direkt ins Display Daten einzugeben. Außerdem berechnet dieser Taschenrechner auch Serienrechnungen wie  $1+2+3+4+5+6+7+8+9$  und Klammerausdrücke wie  $3*(2+4)$ .



19.3: Der Taschenrechner erlaubt umfangreiche Rechenoperationen.

Anhand dieses Beispiels können Sie studieren, wie man Formulareingaben mit JavaScript zur direkten Interaktion verwenden kann. Diese Art Einsatz von JavaScript gehört zu den sinnvollsten Einsatzmöglichkeiten.



#### Quelltext mit Erläuterungen

Das Beispiel zeigt die vollständige HTML-Datei mit dem Taschenrechner.

```
<html>
<head>
<title>Taschenrechner</title>
<script type="text/javascript">
<!--
function Check(Eingabe) {
    var nur_das ="0123456789[()]+*%/";
    for (var i = 0; i < Eingabe.length; i++)
        if (nur_das.indexOf(Eingabe.charAt(i))<0 ) return false;
    return true;
}

function Ergebnis() {
    var x = 0;
```

```

    if (Check(window.document.Rechner.Display.value))
        x = eval(window.document.Rechner.Display.value);
    window.document.Rechner.Display.value = x;
}

function Hinzufuegen(Zeichen) {
    window.document.Rechner.Display.value =
        window.document.Rechner.Display.value + Zeichen;
}

function Sonderfunktion(Funktion) {
    if (Check(window.document.Rechner.Display.value)) {
        if(Funktion == "sqrt") {
            var x = 0;
            x = eval(window.document.Rechner.Display.value);
            window.document.Rechner.Display.value = Math.sqrt(x);
        }
        if(Funktion == "pow") {
            var x = 0;
            x = eval(window.document.Rechner.Display.value);
            window.document.Rechner.Display.value = x * x;
        }
        if(Funktion == "log") {
            var x = 0;
            x = eval(window.document.Rechner.Display.value);
            window.document.Rechner.Display.value = Math.log(x);
        }
    } else window.document.Rechner.Display.value = 0
}
//-->
</script>
<style type="text/css">
<!--
.button { width:60px; text-align:center;
          font-family:System,sans-serif;
          font-size:100%; }
.display { width:100%; text-align:right;
           font-family:System,sans-serif;
           font-size:100%; }
-->
</style>
</head>
<body bgcolor="#FFFFE0" text="#000000">

<form name="Rechner" action="" onSubmit="Ergebnis();return false;">
<table border="5" cellpadding="10" cellspacing="0">
<tr>

```

```

<td bgcolor="#C0C0C0">
<input type="text" name="Display" align="right" class="display"></td>
</tr><tr>
<td bgcolor="#E0E0E0">
<table border="0" cellpadding="0" cellspacing="2">
<tr>
<td><input type="button" width="60" class="button" value=" 7 "
onClick="Hinzufuegen('7')"></td>
<td><input type="button" width="60" class="button" value=" 8 "
onClick="Hinzufuegen('8')"></td>
<td><input type="button" width="60" class="button" value=" 9 "
onClick="Hinzufuegen('9')"></td>
<td><input type="button" width="60" class="button" value=" + "
onClick="Hinzufuegen('+')"></td>
</tr>
<tr>
<td><input type="button" width="60" class="button" value=" 4 "
onClick="Hinzufuegen('4')"></td>
<td><input type="button" width="60" class="button" value=" 5 "
onClick="Hinzufuegen('5')"></td>
<td><input type="button" width="60" class="button" value=" 6 "
onClick="Hinzufuegen('6')"></td>
<td><input type="button" width="60" class="button" value=" - "
onClick="Hinzufuegen('-')"></td>
</tr>
<tr>
<td><input type="button" width="60" class="button" value=" 1 "
onClick="Hinzufuegen('1')"></td>
<td><input type="button" width="60" class="button" value=" 2 "
onClick="Hinzufuegen('2')"></td>
<td><input type="button" width="60" class="button" value=" 3 "
onClick="Hinzufuegen('3')"></td>
<td><input type="button" width="60" class="button" value=" * "
onClick="Hinzufuegen('*')"></td>
</tr>
<tr>
<td><input type="button" width="60" class="button" value=" = "
onClick="Ergebnis()"></td>
<td><input type="button" width="60" class="button" value=" 0 "
onClick="Hinzufuegen('0')"></td>
<td><input type="button" width="60" class="button" value=" . "
onClick="Hinzufuegen('.')"></td>
<td><input type="button" width="60" class="button" value=" / "
onClick="Hinzufuegen('/')"></td>
</tr>
<tr>

```

```

<td><input type="button" width="60" class="button" value="sqrt "
onClick="Sonderfunktion('sqrt')"></td>
<td><input type="button" width="60" class="button" value=" pow "
onClick="Sonderfunktion('pow')"></td>
<td><input type="button" width="60" class="button" value=" log "
onClick="Sonderfunktion('log')"></td>
<td><input type="reset" width="60" class="button" value=" C " ></td>
</tr>
</table>
</td></tr></table>
</form>

</body>
</html>

```

### ***Funktion zur Berechnung des Inhalts des Rechenfelds***

Im Beispiel werden im Kopf der HTML-Datei mehrere Funktionen notiert, die die Funktionalität des Taschenrechners ermöglichen. Die Funktion `Ergebnis()` errechnet den aktuellen Inhalt des Rechenfeldes. Innerhalb der Funktion wird eine Variable `x` definiert, in der das Ergebnis der Berechnung gespeichert wird. Um das Ergebnis zu erhalten, wird die Rechenoperation, die in dem übergebenen Formularfeld (dem Rechenfeld des Taschenrechners) steht, an die mächtige objektunabhängige JavaScript-Funktion `eval()` übergeben. Diese Funktion übernimmt automatisch die gesamte Berechnung der Rechenoperation. Voraussetzung ist natürlich, dass die Rechenoperation keine ungültigen Zeichen enthält. Ansonsten erzeugt JavaScript eine Fehlermeldung.

Aus diesem Grund wird vor jeder Verwendung von `eval()` die Funktion `check()` aufgerufen, welche die Gültigkeit der Eingaben prüft. Nur wenn diese Funktion `true` zurückgibt, wird die Methode `eval()` angewendet. Ansonsten wird im Display der Wert 0 angezeigt.

Um das errechnete Ergebnis, das in der Variablen `x` gespeichert wird, im Rechenfeld des Taschenrechners anzuzeigen, wird dem entsprechenden Formularfeld einfach der Wert von `x` zugewiesen.

### ***Funktion zur Prüfung der Gültigkeit der Eingabe***

Die Funktion `Check(Eingabe)` überprüft, ob die Werte von `Eingabe` nur die vorgegebenen Zeichen enthält. Diese sind in der Variablen `nur_das` abgelegt. Mit einer `for`-Schleife wird die übergebene Zeichenkette Zeichen für Zeichen daraufhin überprüft, ob dieses Zeichen in dem Bereich von `nur_das` vorkommt. Mit der Methode `charAt()` wird aus der Variablen `Eingabe` das Zeichen an der Position `i` extrahiert und an `nur_das.indexOf()` als Parameter übergeben. Die Methode `indexOf()` prüft daraufhin, ob das Zeichen in `nur_das` enthalten ist. Tritt ein ungültiges Zeichen auf, so wird die Funktion abgebrochen und der Wert `false` zurückgegeben. Andernfalls gibt die Funktion `true` zurück.



### ***Funktion zum Hinzufügen eines Zeichens in das Rechenfeld***

Die Funktion `Hinzufuegen()` im Beispiel wird aufgerufen, wenn der Anwender auf einen der Buttons im Taschenrechner-Formular klickt. Das können Ziffern (0 bis 9) oder Rechenoperatoren (+ - \* /) oder ein Dezimalpunkt sein. Um das Zeichen hinzuzufügen, wird dem aktuellen Wert des Rechenfeldes einfach der bisherige Wert plus das neue Zeichen zugewiesen. Dabei kommt der Operator zur Zeichenkettenverknüpfung (+) zum Einsatz. Das neue Zeichen wird der Funktion als Parameter übergeben.

### ***Funktion zur Berechnung von Sonderfunktionen***

Die Funktion `Sonderfunktion()` im Beispiel ist für die Berechnung von drei mathematischen Funktionen zuständig, für die es im Taschenrechner entsprechende Buttons gibt: die Funktionen »sqrt« (Quadratwurzel), »pow« (Quadrat) und »log« (natürlicher Logarithmus über 10). Die Funktion erwartet drei Parameter: den Index des Formulars, in dem sich das Rechenfeld befindet, den Index des Feldes innerhalb dieses Formulars, und eine Zeichenkette, die die auszuführende mathematische Funktion kennzeichnet. Innerhalb der Funktion wird der dritte Parameter abgefragt. Erlaubt sind die Übergabewerte »sqrt«, »pow« und »log«. Falls einer dieser Werte übergeben wurde, berechnet die Funktion innerhalb des entsprechenden If-Zweigs den gewünschten Wert. Dazu wird die Rechenoperation oder die Zahl, die in dem übergebenen Formularfeld (dem Rechenfeld des Taschenrechners) steht, an die JavaScript-Standardfunktion `eval()` übergeben. Diese Funktion übernimmt automatisch die gesamte Berechnung der Rechenoperation. Das Ergebnis wird in `x` gespeichert. Falls es sich nur um eine einfache Zahl handelt, wird diese Zahl in `x` gespeichert. Mit einer entsprechenden JavaScript-Methode, z.B. der Methode `sqrt()` zur Errechnung der Quadratwurzel, wird das Ergebnis der mathematischen Sonderfunktion berechnet und zugleich dem Rechen-/Ergebnisfeld des Taschenrechner-Formulars zugewiesen.

### ***Button-Klick als Zeichen in Rechenfeld eintragen***

Innerhalb des Taschenrechner-Formulars im Beispiel werden die Buttons für Ziffern, Operatoren usw. definiert. Beim Klicken auf einen solchen Button soll natürlich eine entsprechende Reaktion erfolgen. Dies geschieht mit Hilfe des Event-Handlers `onClick=`. Der Event-Handler wird aktiv, wenn der entsprechende Button angeklickt wird, und ruft dann eine der im Dateikopf definierten Funktionen auf. Bei einer Ziffer wird beispielsweise die Funktion `Hinzufuegen()` aufgerufen. Dazu wird der Funktion der erwartete Parameter übergeben, also das hinzuzufügende Zeichen. Das Taschenrechner-Formular erhält bei der Definition einen Namen, im Beispiel den Namen »Rechner«. Auch das Rechen-/Ergebnisfeld, das im JavaScript-Code sehr oft angesprochen wird, erhält einen Namen – den Namen »Display«. Das Formular-Objekt und das Formularelement-Objekt des Rechen-/Ergebnisfeldes kann deshalb mit `window.document.Rechner.Display` adressiert werden.

### ***Inhalt des Rechenfelds löschen***

Der Button mit der Aufschrift »C« im Beispiel dient zum Löschen der bisherigen Eingaben. Dazu braucht man kein JavaScript. Stattdessen genügt es, einen Button vom Typ `reset` zu definieren.

### Absenden des Formulars verhindern

Da der Taschenrechner ein einzeliges Formularfeld enthält, kann dieses Formular jederzeit mit Betätigung der Eingabetaste abgesendet werden. Ein Absenden ist im Zusammenhang mit dem Taschenrechner jedoch unerwünscht. Deshalb wird im einleitenden `<form>`-Tag der Event-Handler `onSubmit` notiert. Dahinter folgt der Aufruf der Funktion `Ergebnis()` und anschließend `return false`. Diese Notation bewirkt, dass wenn der Anwender die Eingabetaste drückt, die Berechnung des Ergebnisses erfolgt und gleichzeitig ein Abschicken des Formulars verhindert wird.

**Beachten Sie:** Im HTML-Standard ist die Verwendung des Attributes `width=` für `input`-Felder nicht erlaubt. Der Netscape 4.x interpretiert jedoch keine CSS-Eigenschaften für solche Formularfelder. Im Interesse des Layouts wurden daher trotz der Verletzung des HTML-Standards die `width`-Attribute verwendet.

## 19.4 Monatskalender

Das Beispiel schreibt in eine HTML-Datei an einer gewünschten Stelle einen frei formatierbaren Kalender des aktuellen Monats. Der aktuelle Tag wird darin hervorgehoben dargestellt. Ein kleines Schmuckstück für Web-Seiten.

Anhand des Beispiels können Sie den praktischen Umgang mit dem `Date`-Objekt von JavaScript studieren, aber ebenso, wie sich durch kontrollierte `document.write`-Ausgaben komplexer HTML-Code erzeugen lässt.



19.4: Der Monatskalender mit dem aktuellen Tag in besonderer Formatierung.



### Quelltext mit Erläuterungen

Das Beispiel zeigt eine vollständige HTML-Datei, in der das JavaScript für den Monatskalender eingebunden ist.

```

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

<h1 style="font-family:Verdana,Arial; font-weight:normal">Monatskalender</h1>

<script type="text/javascript">
<!--
var d = new Date();
var dm = d.getMonth() + 1;
var dj = d.getYear();
if(dj < 999) dj+=1900;
Kalender(dm,dj);

function Kalender(Monat,Jahr) {
Monatsname = new Array
("Januar","Februar","M&auml;rzt","April","Mai","Juni","Juli",
"August","September","Oktober","November","Dezember");
Tag = new Array ("Mo","Di","Mi","Do","Fr","Sa","So");

var KSchrArt = "Verdana,Arial"; /* Schriftart Kalenderkopf */
var KSchrGroesse = 3;          /* Schriftgroesse 1-7 Kalenderkopf */
var KSchrFarbe = "#FFFF00";    /* Schriftfarbe Kalenderkopf */
var Khgrund = "#000066";       /* Hintergrundfarbe Kalenderkopf */
var TSchrArt = "Verdana,Arial"; /* Schriftart Tagesanzeige */
var TSchrGroesse = 3;          /* Schriftgroesse 1-7 Tagesanzeige */
var TSchrFarbe = "#000000";    /* Schriftfarbe Tagesanzeige */
var Thgrund = "#D0F0F0";       /* Hintergrundfarbe Tagesanzeige */
var SoFarbe = "#E00000";       /* Schriftfarbe f. Sonntage */
var Ahgrund = "#FFFF00";       /* Hintergrundfarbe f. heutigen Tag */

var jetzt = new Date();
var DieserMonat = jetzt.getMonth() + 1;
var DiesesJahr = jetzt.getYear();
if(DiesesJahr < 999) DiesesJahr+=1900;
var DieserTag = jetzt.getDate();
var Zeit = new Date(Jahr,Monat-1,1);
var Start = Zeit.getDay();
if(Start > 0) Start--;
else Start = 6;
var Stop = 31;
if(Monat==4 ||Monat==6 || Monat==9 || Monat==11 ) --Stop;
if(Monat==2) {
    Stop = Stop - 3;
    if(Jahr%4==0) Stop++;

```

```

if(Jahr%100==0) Stop--;
if(Jahr%400==0) Stop++;
}
document.write('<table border="3" cellpadding="1" cellspacing="1">');
var Monatskopf = Monatsname[Monat-1] + " " + Jahr;
SchreibeKopf(Monatskopf,Khgrund,KSchrFarbe,KSchrGroesse,KSchrArt);
var Tageszahl = 1;
for(var i=0;i<=5;i++) {
    document.write("<tr>");
    for(var j=0;j<=5;j++) {
        if((i==0)&&(j < Start))
            SchreibeZelle("&#160;",Thgrund,TSchrFarbe,TSchrGroesse,TSchrArt);
        else {
            if(Tageszahl > Stop)
                SchreibeZelle("&#160;",Thgrund,TSchrFarbe,TSchrGroesse,TSchrArt);
            else {
                if((Jahr==DiesesJahr)&&(Monat==DieserMonat)&&(Tageszahl==DieserTag))
                    SchreibeZelle(Tageszahl,Ahgrund,TSchrFarbe,TSchrGroesse,TSchrArt);
                else
                    SchreibeZelle(Tageszahl,Thgrund,TSchrFarbe,TSchrGroesse,TSchrArt);
                Tageszahl++;
            }
        }
    }
}
if(Tageszahl > Stop)
    SchreibeZelle("&#160;",Thgrund,SoFarbe,TSchrGroesse,TSchrArt);
else {
    if((Jahr==DiesesJahr)&&(Monat==DieserMonat)&&(Tageszahl==DieserTag))
        SchreibeZelle(Tageszahl,Ahgrund,SoFarbe,TSchrGroesse,TSchrArt);
    else
        SchreibeZelle(Tageszahl,Thgrund,SoFarbe,TSchrGroesse,TSchrArt);
    Tageszahl++;
}
document.write("<\/tr>");
}
document.write("<\/table>");
}

function SchreibeKopf(Monatstitel,HgFarbe,SchrFarbe,SchrGroesse,SchrArt) {
document.write("<tr>");
document.write('<td align="center" colspan="7" valign="middle"
bgcolor="'+HgFarbe+'">');
document.write('<font size="'+SchrGroesse+'" color="'+SchrFarbe+'"
face="'+SchrArt+'"><b>');
document.write(Monatstitel);
document.write("<\/b><\/font><\/td><\/tr>");
document.write("<tr>");

```

```

for(var i=0;i<=6;i++)
    SchreibeZelle(Tag[i],HgFarbe,SchrFarbe,SchrGroesse,SchrArt);
document.write("<\tr>");
}

function SchreibeZelle(Inhalt,HgFarbe,SchrFarbe,SchrGroesse,SchrArt) {
document.write('<td align="center" valign="middle" bgcolor="'+HgFarbe+'">');
document.write('<font size="'+SchrGroesse+'" color="'+SchrFarbe+'"
face="'+SchrArt+'"><b>');
document.write(Inhalt);
document.write("<\b><\font><\td>");
}
//-->
</script>

</body>
</html>

```

### Aktuelles Datum ermitteln

Im Script-Bereich wird zunächst mit `d = new Date()` ein neues Datumsobjekt mit dem aktuellen Zeitpunkt erzeugt. In den Variablen `dm` und `dj` werden anschließend aus dem neuen Datumsobjekt `d` der Monat und das Jahr ermittelt. Mit diesen beiden Variablen wird die Funktion `Kalender()` aufgerufen.

Die Anweisung `if(dj < 999) dj+=1900;` ist dabei nötig, um einigen älteren Browsern, die bei dem Aufruf der `getYear`-Methode nicht Jahr-2000-kompatibel reagieren, zum Ermitteln des gewünschten Jahres zu verhelfen.

### Funktion »Kalender()«

Die Funktion `Kalender()` ist als in sich abgeschlossenes Unterprogramm realisiert, das einen Monat und ein Jahr als Parameter erwartet und als Monatskalender angezeigt werden soll. Auf diese Weise lässt sich die Funktion auch noch anderweitig nutzen als zur einmaligen Ausgabe des aktuellen Kalendermonats.

Zu Beginn der Funktion `Kalender()` werden erst einmal verschiedene Variablen definiert. Dazu gehören die gewünschten Monatsnamen und die Wochentage, die jeweils als Array-Objekte angelegt werden. Wenn Sie andere, zum Beispiel landesspezifische Namen wie »Jänner« wünschen, ändern Sie den entsprechenden Namen einfach.

Zu den Variablen, die am Beginn von `Kalender()` definiert werden, gehören auch die Angaben zu Farben, Schriftarten und Schriftgrößen für Kalenderhintergrund und Anzeige der Tage. Die Variablendefinitionen sind im Beispiel zum besseren Verständnis auskommentiert. Ändern Sie diese Werte, wenn Sie den Kalender verwenden möchten, nach Ihren Wünschen.

### **Ermitteln, ob aktiver Tag im anzuzeigenden Monat liegt**

Da die Funktion `Kalender()` ja als Parameter übergeben bekommt, welchen Monat in welchem Jahr sie ausgeben soll, kann es sich durchaus um einen anderen als den aktuellen Monat handeln (nur im obigen Beispiel wird die Funktion mit dem aktuellen Monat/Jahr aufgerufen). Die Funktion ermittelt deshalb noch einmal separat die Daten des aktuellen Tages (`DieserTag`), des aktuellen Monats (`DieserMonat`) und des aktuellen Jahres (`DiesesJahr`). Die entsprechenden Daten werden später benötigt, um den aktuellen Tag farblich hervorzuheben, sofern er in dem auszugebenden Monat vorkommt. Auch dabei wird wieder die Anpassung für das Jahr 2000 durchgeführt.

### **Anzeige in Tabelle vorbereiten**

Der Monatskalender soll mit Hilfe von `document.write()`-Anweisungen in eine HTML-Tabelle geschrieben werden. Zunächst muss jedoch ermittelt werden, wie viele Tage der auszugebende Monat hat und an welchem Wochentag er beginnt. Davon hängt ab, wie die Tage in die Tabelle geschrieben werden.

In der Variablen `Start` wird der Wochentag gespeichert, auf den der erste Tag des auszugebenden Monats fällt. Dazu wird in einer Variablen `Zeit` ein Datumsobjekt erzeugt, das sich mit `Date(Jahr,Monat-1,1)` auf den ersten Tag des auszugebenden Monats bezieht. Mit `Zeit.getDay()` wird der Wochentag dieses Tages ermittelt (0 für Montag bis 6 für Sonntag).

In der Variablen `Stop` wird die Anzahl der Tage ermittelt, die der Monat hat. Wenn es einer der Monate 4, 6, 9 oder 11 ist, hat der Monat 30 Tage. Wenn es der Monat 2 ist, ist etwas Kalenderakrobatik erforderlich. Denn dieser Monat kann 28 oder 29 Tage haben, abhängig einmal davon, ob es ein Schaltjahr ist (durch 4 teilbar), ob es trotzdem kein Schaltjahr ist (durch 100 teilbar), oder ob es doch wieder ein Schaltjahr ist (durch 400 teilbar).

### **Tabelle erzeugen**

Nachdem alle Vorbereitungen getroffen sind, kann die Tabelle mit dem Kalendermonat geschrieben werden. Dies geschieht mit Hilfe von `document.write()`-Befehlen, die dynamisch die entsprechenden HTML-Konstrukte in die Datei schreiben. Die Funktion `Kalender()` selbst schreibt jedoch nur den Anfang und das Ende der Tabelle. Ansonsten übernimmt sie nur die Kontrolle, was genau in welcher Reihenfolge geschrieben werden soll. Um den Tabellenkopf mit Monats- und Jahresangabe sowie den Wochentagnamen zu schreiben, ruft sie die Unterfunktion `SchreibeKopf()` auf. Um die einzelnen Datenzellen zu schreiben, wird jeweils die Unterfunktion `SchreibeZelle()` aufgerufen. Um Letztere mit den richtigen Daten zu versorgen, muss die `Kalender()`-Funktion die zuvor ermittelten Angaben zum Wochentag des ersten Tages im Monat sowie das aktuelle Tagesdatum berücksichtigen und die entsprechenden Parameter korrekt übergeben. Falls eine Zelle leer bleiben soll, wird das erzwungene Leerzeichen `&#160;` übergeben, denn nur so wird die Tabellenzelle nachher sichtbar dargestellt, auch wenn sie keinen Inhalt hat. Wenn die Zelle eine Tageszahl anzeigen soll, wird eine entsprechende Variable übergeben.

Die Funktionen `SchreibeKopf()` und `SchreibeZelle()` schreiben die einzelnen Datenzellen und setzen dabei die Variablen zur Gestaltung des Kalenders ein.

## 19.5 Persönliche Seitenbesuche zählen mit Cookies

Cookies (Kekse) sind ein nicht unumstrittenes Feature, das Netscape in JavaScript eingebaut hat. Cookies erlauben einem JavaScript-Programm, Daten auf der Festplatte des Anwenders zu speichern. Dies geschieht jedoch in kontrollierter Form. Ein »Ausspionieren« der Anwenderfestplatte ist dabei ebenso wenig möglich wie das Platzieren von ausführbarem Code, also etwa Viren. Denn wenn Sie in einem JavaScript einen Cookie schreiben, können Sie nicht angeben, wohin der Cookie beim Anwender gespeichert wird. Das kontrolliert der WWW-Browser des Anwenders. Außerdem können Cookies keine unkontrollierten Datenmengen auf den Anwenderrechner schreiben, sondern nur eine begrenzte Anzahl von Zeilen. Jede dieser Zeilen definiert eine Variable und weist dieser Variablen einen Wert zu. Ein Cookie lässt sich also mit einem Eintrag in einer Konfigurationsdatei vergleichen – mit dem Unterschied, dass der Cookie keine Konfigurationsdaten des Anwenderrechners ansprechen kann.

Durch die Möglichkeit, Daten zu speichern, werden jedoch viele neue Anwendungsbereiche für JavaScript möglich. So auch das hier beschriebene Beispiel. Denn um zeitlich auseinander liegende Seitenbesuche zu zählen, muss man die Anzahl der bisherigen Seitenbesuche irgendwo fest speichern, beim nächsten Seitenbesuch wieder auslesen, den Zähler um 1 erhöhen und den neuen Wert wieder speichern. Genau das tut das hier beschriebene Beispiel.

**Beachten Sie:** Das Zählen von persönlichen Seitenbesuchen ist etwas anderes als das Zählen aller Seitenbesuche. Um alle Zugriffe auf eine Seite zu verwalten, muss das Programm seine Zählerstände auf dem Server-Rechner ablegen. Deshalb funktioniert ein klassischer »Counter« nur mit CGI. Das hier beschriebene Script zählt nur, wie oft ein bestimmter Anwender eine bestimmte Seite besucht.



19.5: Der Zähler mit einem Cookie zeigt die bisher erfolgten Besuche an.



### Quelltext mit Erläuterungen

Das Beispiel zeigt eine vollständige HTML-Datei. Der Zähler gilt für diese eine Datei.

```
<html>
<head>
<title>Persönliche Seitenbesuche zählen mit Cookies</title>
<script type="text/javascript">
<!--
function Wortholen() {
```

```

var Wert = "";
if(document.cookie) {
    var Wertstart = document.cookie.indexOf("=") + 1;
    var Wertende = document.cookie.indexOf(";");
    if (Wertende == -1)
        Wertende = document.cookie.length;
    Wert = document.cookie.substring(Wertstart,Wertende);
}
return Wert;
}

function WertSetzen(Bezeichner, Wert, Verfall) {
    var jetzt = new Date();
    var Auszeit = new Date(jetzt.getTime() + Verfall);
    document.cookie = Bezeichner+"="+Wert+"; expires="+Auszeit.toGMTString()+"";
}

function Zaehlerstand() {
    var Verfallszeit = 1000*60*60*24*365;
    var Anzahl = WertHolen();
    var Zaehler = 0;
    if(Anzahl != "") Zaehler = parseInt(Anzahl);
    if(document.cookie) {
        Zaehler = Zaehler + 1;
        WertSetzen("Zaehler",Zaehler,Verfallszeit);
    }
    else {
        Zaehler = 1;
        WertSetzen("Zaehler",Zaehler,Verfallszeit);
    }
    return(Zaehler);
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">

<h1>Eine Seite, die etwas über Sie weiß;...</h1>

<script type="text/javascript">
<!--
    var x = Zaehlerstand();
    document.write("<p>Dies ist Ihr <b>" + x + ".</b> Besuch auf dieser
Seite!</p>");
//-->
</script>

```



```
<p>Benutzen Sie doch mal die Reload-Funktion Ihres Browsers ...</p>
</body>
</html>
```

### **Cookie lesen und gespeicherten Wert ermitteln**

Die Funktion `WertHolen()`, die im Beispiel definiert ist, ist allgemein geschrieben. Sie können diese Funktion nutzen, um einen Cookie auszulesen. Wenn die Datei schon einmal einen Cookie geschrieben hat, wird dessen Wert ermittelt und zurückgegeben. Wenn noch kein Cookie existiert, wird eine leere Zeichenkette `" "` zurückgegeben.

Mit der Abfrage `if(document.cookie)` ermittelt die Funktion, ob bereits ein Cookie vorhanden ist. Wenn ja, wird dessen Wert ermittelt. Ein Cookie besteht normalerweise aus einem Bezeichner und einem Wert, zum Beispiel: `Zaehler=4`. Dabei ist `Zaehler` der Bezeichner und `4` der Wert, der diesem Bezeichner zugewiesen ist. Die Funktion `WertHolen()` holt sich den Wert, indem sie mit `indexOf()`-Befehlen den Anfang und das Ende des Wertes innerhalb des gespeicherten Cookies ermittelt. Diesen Wert weist sie der Variablen `Wert` zu. Die Variable `Wert` wird am Ende der Funktion zurückgegeben.

### **Cookie schreiben und Wert setzen**

Die Funktion `WertSetzen()`, die im Beispiel definiert ist, ist ebenfalls allgemein geschrieben. Diese Funktion erwartet drei Parameter:

- `Bezeichner` ist das Element vor dem Gleichheitszeichen, zum Beispiel `Zaehler`.
- `Wert` ist der Wert, der dem Bezeichner zugewiesen und mit gespeichert werden soll.
- `Verfall` ist eine Zahl, die als Anzahl Millisekunden interpretiert wird. Jeder Cookie, der über die aktuelle Browser-Sitzung hinaus gespeichert bleiben soll, muss eine Angabe dazu enthalten, wann der Browser den Cookie löschen soll. Das soll verhindern, dass Cookies »ewig« gespeichert bleiben. Der Verfallszeitpunkt des Cookies muss in Form des GMT-Zeitformats angegeben werden. Darum brauchen Sie sich aber nicht zu kümmern, da die Funktion `WertSetzen()` dies automatisch erledigt. Die Funktion ermittelt zu diesem Zweck die aktuelle Zeit (`var jetzt = new Date();`). In der Variablen `Auszeit` errechnet sie einen zukünftigen Zeitpunkt aus dem übergebenen Parameter `Verfall` und dem ermittelten aktuellen Zeitpunkt.

Mit der Anweisung, die mit `document.cookie=` beginnt, wird der Cookie geschrieben.

### **Übergeordnete Funktion zum Verwalten des Cookies**

Die beiden allgemeinen Funktionen `WertHolen()` und `WertSetzen()` benötigen nun noch ein paar allgemeine Anweisungen, die den Cookie verwalten. Dies geschieht im Beispiel in der Funktion `Zaehlerstand()`. Zunächst definiert diese Funktion einen Verfallswert, der später beim Schreiben des Cookies an die Funktion `WertSetzen()` übergeben wird. Mit dem Rechenausdruck `1000*60*60*24*365` wird die Millisekundenzahl eines Jahres ermittelt. Der Cookie soll also ein Jahr lang gespeichert werden.

Um die persönlichen Seitenbesuche des Anwenders zu verwalten, muss die Funktion `Zaehlerstand()` Folgendes tun: Sie muss erstens den aktuellen Zählerstand ermitteln, zweitens den Zähler um 1 erhöhen und drittens den neuen Zählerstand wieder speichern. Wenn noch kein Cookie vorhanden ist, d.h. wenn der Anwender die Seite zum ersten Mal besucht, muss der Zähler auf 1 gesetzt und gespeichert werden.

Zuerst ruft die Funktion `Zaehlerstand()` deshalb die Funktion `WertHolen()` auf, um den aktuellen Zählerstand zu ermitteln. Der Rückgabewert von `WertHolen()` wird in der Variablen `Anzahl` gespeichert. Wenn noch kein Cookie vorhanden ist, steht in `Anzahl` anschließend eine leere Zeichenkette `""`. Ist ein Wert vorhanden, enthält `Anzahl` den Wert in Form einer Zeichenkette. Da man mit Zeichenketten nicht rechnen kann, zum Hochzählen des Zählers jedoch ein numerischer Wert benötigt wird, wird im Beispiel noch eine numerische Variable namens `Zaehler` definiert. Mit der `parseInt()`-Methode wird dieser Variablen der Wert von `Anzahl` als `Zahl` zugewiesen.

Wenn ein Cookie vorhanden ist, wird der Zähler um 1 erhöht. Anschließend wird die Funktion `WertSetzen()` aufgerufen, die den Cookie speichert. Als Bezeichner wird das Wort `"Zaehler"` übergeben, als Wert der Inhalt der Variablen `Zaehler` und als Verfallszeitpunkt der Inhalt der Variablen `Verfallszeit`.

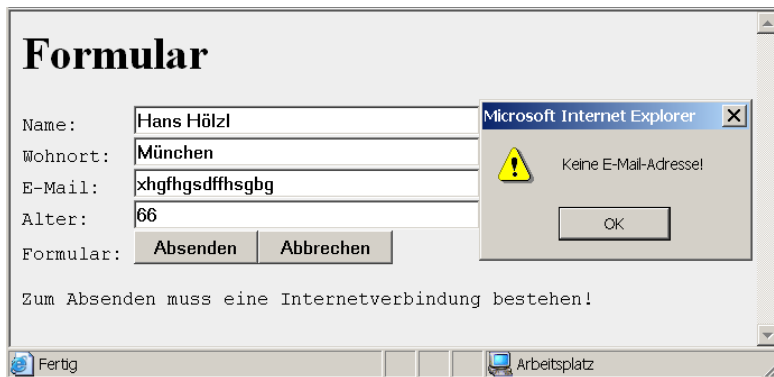
Wenn kein Cookie vorhanden ist, wird die Funktion `WertSetzen()` genauso aufgerufen. Vorher wird jedoch der Inhalt der Variablen `Zaehler` auf 1 gesetzt.

Am Ende gibt die Funktion `Zaehlerstand()` mit Hilfe von `return()` zurück. Damit kann diese Funktion beispielsweise an anderer Stelle für eine Ausgabe innerhalb der Seite aufgerufen werden, so wie im Beispiel. Dort ist im `body`-Bereich ein weiterer JavaScript-Bereich notiert. Innerhalb davon wird die Funktion `Zaehlerstand()` aufgerufen, und ihr Rückgabewert, also der aktuelle Zähler, wird in der Variablen `x` gespeichert. Anschließend wird mit `document.write` HTML-Code ins Anzeigefenster des Browsers geschrieben, wobei die Variable `x` mit ausgegeben wird.

## 19.6 Formulareingaben überprüfen

Wenn Sie HTML-basierte Formulare im WWW anbieten, kann jeder Anwender irgendeinen Unsinn in das Formular schreiben und es dann abschicken. Betroffen davon sind vor allem Eingabefelder. Dadurch erhalten Sie überflüssige E-Mails, oder ein CGI-Script, das die Formulareingaben automatisch verarbeitet, speichert leere oder nutzlose Daten ab. Mit Hilfe von JavaScript können Sie Formulareingaben überprüfen, bevor das Formular abgeschickt wird. Bei fehlenden oder offensichtlich fehlerhaften Eingaben können Sie das Absenden des Formulars verhindern.

Das Beispiel auf dieser Seite zeigt, wie eine solche Überprüfung im Prinzip funktioniert. Je nachdem, was Ihr Formular enthält, müssen Sie das Beispiel natürlich erweitern.



19.6: Das JavaScript bemängelt die eingetippte E-Mail-Adresse.

**Quelltext mit Erläuterungen**

Das Beispiel zeigt die vollständige HTML-Datei mit dem Formular und dem JavaScript zum Überprüfen der Formulareingaben. Das Beispielformular ruft nach erfolgreichem Absenden ein CGI-Script auf dem Server von *teamone.de* auf.

```
<html>
<head>
<title>Formulareingaben &uuml;berpr&uuml;fen</title>
<script type="text/javascript">
<!--
function chkFormular()
{
  if(document.Formular.User.value == "") {
    alert("Bitte Ihren Namen eingeben!");
    document.Formular.User.focus();
    return false;
  }
  if(document.Formular.Ort.value == "") {
    alert("Bitte Ihren Wohnort eingeben!");
    document.Formular.Ort.focus();
    return false;
  }
  if(document.Formular.Mail.value == "") {
    alert("Bitte Ihre E-Mail-Adresse eingeben!");
    document.Formular.Mail.focus();
    return false;
  }
  if(document.Formular.Mail.value.indexOf('@') == -1) {
    alert("Keine E-Mail-Adresse!");
    document.Formular.Mail.focus();
    return false;
  }
}
```

```

if(document.Formular.Alter.value == "") {
    alert("Bitte Ihr Alter eingeben!");
    document.Formular.Alter.focus();
    return false;
}
var chkZ = 1;
for(i=0;i<document.Formular.Alter.value.length;++i)
    if(document.Formular.Alter.value.charAt(i) < "0"
    || document.Formular.Alter.value.charAt(i) > "9")
        chkZ = -1;
if(chkZ == -1) {
    alert("Altersangabe keine Zahl!");
    document.Formular.Alter.focus();
    return false;
}
}
//-->
</script>
</head>
<body bgcolor="#EEEEEE" text="#000000">

<h1>Formular</h1>

<form name="Formular" action="http://selfaktuell.teamone.de/cgi-bin/formview.pl"
method="post" onSubmit="return chkFormular()">
<pre>
Name:      <input type="text" size="40" name="User">
Wohnort:   <input type="text" size="40" name="Ort">
E-Mail:    <input type="text" size="40" name="Mail">
Alter:     <input type="text" size="40" name="Alter">
Formular:  <input type="submit" value="Absenden"><input type="reset"
value="Abbrechen">

Zum Absenden muss eine Internet-Verbindung bestehen!
</pre>
</form>

</body>
</html>

```

### Formular definieren

Im Beispiel wird ein gewöhnliches HTML-Formular mit einigen Eingabefeldern, einem Reset-Button zum Zurücksetzen des Formulars und einem Submit-Button zum Absenden des Formulars definiert. Die einzige Besonderheit ist die Anweisung `onSubmit="return chkFormular()"` im einleitenden `<form>`-Tag. Der Event-Handler `onSubmit=` wird aktiv, wenn der Anwender auf den Submit-Button klickt, um das Formular abzusenden. Dann wird die JavaScript-Funktion

`chkFormular()` aufgerufen, die im Dateikopf in einem JavaScript-Bereich notiert ist. Wenn die Funktion Fehler bei den Eingaben findet, gibt sie den Wert `false` (*falsch*) zurück, andernfalls den voreingestellten Wert `true` (*wahr*). Mit `return` wird dieser Wert im einleitenden `<form>`-Tag an den Browser weitergegeben. Der Browser kennt die beiden Werte `true` und `false`. Im Zusammenhang mit `onSubmit=` reagiert er so, dass er das Formular nur dann abschickt, wenn der Wert `true` ist. Ist der Wert `false`, wird das Absenden der Formulardaten verhindert.

### Formular überprüfen

Die eigentliche Formularüberprüfung findet im Beispiel in der Funktion `chkFormular()` statt, die in dem Script-Bereich im Dateikopf notiert ist. Im Beispiel soll jedes Eingabefeld daraufhin überprüft werden, ob es einen Inhalt hat, d.h. ob der Anwender es ausgefüllt hat. Bei der Angabe zur E-Mail-Adresse soll zusätzlich überprüft werden, ob in der Eingabe das Zeichen `@` vorkommt. Ist das nicht der Fall, kann man davon ausgehen, dass der Anwender in dem entsprechenden Feld Unsinn eingegeben hat. Bei der Altersangabe schließlich soll zusätzlich überprüft werden, ob die Eingabe eine positive ganze Zahl ist. Andernfalls muss man auch hierbei eine unsinnigen Angabe vermuten.

Mit einer Abfrage wie `if(document.Formular.User.value == "")` wird ermittelt, ob das entsprechende Formularfeld (im Beispiel also das Feld mit dem Namen `User` im Formular mit dem Namen `Formular`) überhaupt einen Inhalt hat. Es wird also auf die leere Zeichenkette `""` hin geprüft.

Mit einer Abfrage wie `if(document.Formular.Mail.value.indexOf('@') == -1)` wird ermittelt, ob in der Eingabe ein bestimmtes Zeichen fehlt (hier das `@`-Zeichen, das für E-Mail-Adressen charakteristisch ist).

Mit einem Ausdruck wie:

```
for(i=0;i<document.Formular.Alter.value.length;++i)
if(document.Formular.Alter.value.charAt(i) < "0"
|| document.Formular.Alter.value.charAt(i) > "9")
```

ermitteln Sie Zeichen für Zeichen, ob ein eingegebener Wert etwas anderes als Ziffern enthält. Wenn ja, wird im Beispiel eine Merkvariable namens `chkZ` auf den Wert `-1` gesetzt.

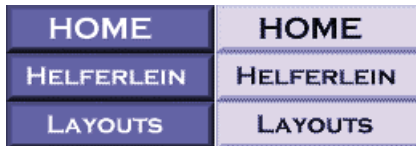
Bei allen Abfragen dieser Art werden jeweils drei Anweisungen ausgeführt, wenn die Anwender-eingaben der Prüfung nicht standhalten. Zuerst wird mit `alert()` jeweils in einem Meldungsfenster ausgegeben, was der Anwender falsch gemacht hat. In der zweiten Anweisung wird mit der `focus()`-Methode auf das Formularelement positioniert, auf das sich der Fehler bezieht. Der Anwender kann seine Feldeingabe also gleich korrigieren. Die dritte Anweisung schließlich gibt `false` zurück. Dadurch wird das Absenden des Formulars verhindert.

## 19.7 Dynamische grafische Buttons

Auf WWW-Seiten kommen häufig grafische Button-Leisten zum Einsatz, die Verweise zu bestimmten Seiten des Projekts enthalten. Wenn der Anwender mit der Maus über solche Grafiken fährt, erkennt er am veränderten Mauszeiger und an den Verweiszielanzeigen in der Statuszeile des WWW-Browsers, dass es sich um Verweise handelt. Mit Hilfe von JavaScript können Sie sol-

che grafischen Verweise jedoch noch deutlicher kenntlich machen und der Benutzerführung gleichzeitig ein wenig mehr Pep verleihen (»Mouseover-Effekt«).

Dazu brauchen Sie je zwei gleichartige, farblich unterschiedliche Grafiken für je einen Grafik-Button. Im Anzeigebeispiel zu diesem Abschnitt sind das die beiden folgenden Button-Paare:



19.7: Drei Buttons im normalen Zustand (links) und gedrückten Zustand (rechts).

Mit Hilfe von JavaScript lässt sich nun eine Grafik durch eine andere ersetzen, zum Beispiel wenn die Grafik als Verweises dient und der Anwender mit der Maus über die Grafik fährt. Wie das im Einzelnen funktioniert, wird im folgenden beschrieben.



19.8: Mit Script ändern die Buttons beim Überfahren mit der Maus die Grafik.

Dieses Beispiel funktioniert mit Netscape ab Version 3.x und mit dem MS Internet Explorer ab Version 4.x. Beachten Sie, dass es bei Verwendung dynamischer Grafiken in Tabellen innerhalb von Tabellen bei Netscape zu Verschiebungsproblemen kommen kann.



### Quelltext mit Erläuterungen

Das Beispiel zeigt eine vollständige HTML-Datei mit Button-Leiste und dem JavaScript zum Steuern der Mousover-Effekte.

```
<html>
<head>
<title>Dynamische grafische Buttons</title>
<script type="text/javascript">
<!--
Normal1 = new Image();
```



```
</td>
</tr>
</table>

</body>
</html>
```

### **Neue Grafikobjekte für Highlight-Grafiken erzeugen**

Für jede Grafik, die Sie dynamisch anzeigen möchten, müssen Sie eine Instanz des image-Objekts erzeugen. Das gilt sowohl für die Grafiken, die zunächst in der HTML-Datei referenziert werden, als auch für diejenigen, die beim Überfahren mit der Maus dynamisch angezeigt werden sollen.

Dazu wird im Beispiel mit Anweisungen wie `Normal1 = new Image();` eine Objektinstanz erzeugt. Nachdem die Objektinstanz erzeugt ist, sind unter dem gewählten Objektnamen, im ersten Fall `Normal1`, alle Eigenschaften des Objekts ansprechbar. Zunächst enthält das Objekt noch gar keine Daten. Mit der Anweisung `Normal1.src = "button1.gif";` wird dem neuen Grafikobjekt eine Grafikdatei zugewiesen.

Wiederholen Sie die beiden beschriebenen Anweisungen für jede Grafikdatei, die von dynamischen Änderungen betroffen sein wird – und zwar sowohl für Grafiken, die im »Normalfall« angezeigt werden, als auch für Grafiken, die beim Darüberfahren mit der Maus dynamisch angezeigt werden. Vergeben Sie dabei jedes Mal einen neuen Objektnamen. Im obigen Beispiel sind die entsprechenden Befehle auskommentiert, sodass Sie erkennen können, was an welcher Stelle definiert wird.

### **Funktion »Bildwechsel()« zum dynamischen Austauschen von Grafiken**

Der bisherige Code im Beispiel wird beim Einlesen der HTML-Datei direkt ausgeführt, da er nicht in einer Funktion gebunden ist. Die Ausführung dieses Codes bewirkt aber keine sichtbaren Ausgaben und wird vom Anwender gar nicht bemerkt. Das, was am Bildschirm mit Hilfe von JavaScript passieren soll, nämlich das dynamische Austauschen eines Bildes, geschieht in der definierten Funktion `Bildwechsel()`.

Die Funktion soll aufgerufen werden, wenn der Anwender mit der Maus über einen grafischen Verweis fährt und wenn er den Verweisbereich mit der Maus wieder verlässt. Dazu benötigt die Funktion zwei Parameter: die wievielte Grafik in der Datei ausgetauscht werden soll (Parameter `Bildnr`), und durch welches zuvor definierte Grafikobjekt das Bild ersetzt werden soll (Parameter `Bildobjekt`). Die Funktion kommt dann mit einer einzigen Anweisung aus. Diese Anweisung ersetzt das vorhandene Bild durch das neue. Beachten Sie hier den wichtigen Zusammenhang: das dynamische Ersetzen einer Grafik ist nur möglich, wenn für die neue Grafik zuvor eine Instanz des Grafikobjekts erzeugt wurde. Im Beispiel geschah dies ja ganz am Anfang des Scripts.

### **Grafische Verweise mit Event-Handlern für Mausbewegungen des Anwenders**

Damit der gewünschte Effekt zustande kommt, müssen Sie in der HTML-Datei Grafiken als Verweise definieren. Im einleitenden Verweis-Tag werden die Event-Handler `onMouseOver=` und `onMouseOut=` notiert. Bei `onMouseOver=` (wenn der Anwender mit der Maus über den verweissensitiven Bereich, hier die Grafik, fährt) wird die im Script-Bereich definierte Funktion



`Bildwechsel()` aufgerufen, ebenso bei `onMouseOut=` (wenn der Mauszeiger den verweissensitiven Bereich wieder verlässt).

Beim Aufruf von `Bildwechsel()` werden jeweils die beiden benötigten Parameter übergeben. Zählen Sie dazu die referenzierten Grafikdateien in der HTML-Datei, aber fangen Sie bei 0 an zu zählen, d.h. 0 für die erste Grafik in der Datei, 1 für die zweite Grafik usw. Übergeben Sie den Indexwert der Grafik, die in den Verweis eingebettet ist. Im Beispiel sind in der gesamten HTML-Datei keine anderen als die dynamisch zu tauschenden Grafiken referenziert. Deshalb wird im ersten Fall 0 (erste Grafik der Datei) übergeben, im zweiten Fall 1 (zweite Grafik der Datei).

Beim zweiten Parameter übergeben Sie den Objektnamen für das gewünschte Grafikobjekt. Das ist einer der Namen, die Sie am Anfang des Script-Bereichs vergeben haben. Im Beispiel wird im ersten Fall z.B. `Highlight1` bei `onMouseover` übergeben, und `Normal1` bei `onMouseout`.

## 19.8 Aufenthaltsdauer auf Web-Seite anzeigen

Das Beispiel auf dieser Seite zeigt in einem kleinen Feld laufend an, wie viele Minuten und Sekunden seit dem Aufruf der Seite verstrichen sind. Gedacht ist die Anzeige des Feldes vor allem für HTML-Dateien, die innerhalb eines Frame-Fensters ständig angezeigt bleiben, etwa in einem Frame-Fenster mit einem Logo oder mit Verweisen. In solchen Fällen läuft die Uhr während der gesamten Besuchszeit, die der Anwender auf den Seiten verbringt. Man mag das als eine überflüssige Spielerei abtun. Aber angenommen, es wird etwas angezeigt, wobei der Anwender schnell die teure Online-Zeit vergisst, etwa ein Spiel oder ein Chat-Fenster, dann kann eine solche kleine »Erinnerung« in Form einer laufenden Zeituhr durchaus nützlich sein.



19.9: Im linken Frame tickt die Uhr und zeigt an, wie lange der Aufenthalt schon dauert.



### Quelltext mit Erläuterungen

Das Beispiel zeigt die HTML-Datei, in der das JavaScript und die Zeitanzeige stehen. Im Anzeigebeispiel ist diese Datei in ein Frameset eingebunden.

```

<html>
<head>
<title>Aufenthaltsdauer auf Web-Seite anzeigen</title>
<script type="text/javascript">
<!--
var Jetzt = new Date();
var Start = Jetzt.getTime();

function ZeitAnzeigen() {
var absSekunden = Math.round(ZeitBerechnen());
var relSekunden = absSekunden % 60;
var absMinuten = Math.abs(Math.round(absSekunden-30/60));
var anzSekunden = "" + ((relSekunden > 9) ? relSekunden : "0" + relSekunden);
var anzMinuten = "" + ((absMinuten > 9) ? absMinuten : "0" + absMinuten);
window.document.Anzeige.Zeit.value = anzMinuten + ":" + anzSekunden;
window.setTimeout('ZeitAnzeigen()',1000);
}

function ZeitBerechnen() {
var Immernoch = new Date();
return((Immernoch.getTime() - Start)/1000);
}
// -->
</script>
</head>
<body bgcolor="#EEEEEE" text="#000000"
onLoad="window.setTimeout('ZeitAnzeigen()',1000)">

<form name="Anzeige" action="">
<input size="7" name="Zeit" value="00:00">
</form>

<p>viel Spa&szlig; im rechten Frame-Fenster!</p>
</body>
</html>

```

### Startzeit ermitteln

Beim Einlesen der Datei wird zunächst der aktuelle Zeitpunkt ermittelt. Dieser Zeitpunkt dient als Startzeit der Anzeige. Mit `Jetzt = new Date()` wird ein neues Datumsobjekt mit dem aktuellen Zeitpunkt in der Variablen `Jetzt` gespeichert. Mit `Jetzt.getTime` wird eine absolute Zahl in Millisekunden ermittelt, die in der Variablen `Start` gespeichert wird.

### Funktion zur Zeitanzeige erstmals aufrufen

Im einleitenden `<body>`-Tag steht im Beispiel der erste Aufruf der Funktion `ZeitAnzeigen()`, die die Zeit anzeigt und die Anzeige jede Sekunde aktualisiert. Der erste Aufruf erfolgt nach einer Sekunde. Dazu wird der Funktionsaufruf in die `setTimeout()`-Methode mit einer Verzögerungszeit von 1000 Millisekunden (1 Sekunde) eingebettet.

Im Dateikörper steht dann ein Formular, das nur aus einem einzigen kleinen Eingabefeld besteht. Dieses Eingabefeld wird als Ausgabefeld für die Zeitanzeige zweckentfremdet. Es wird mit dem Wert 00:00 vorbelegt, damit auch in der ersten Sekunde schon etwas in dem Feld steht.

### **Zeit anzeigen und Anzeige laufend aktualisieren**

Die Funktion `ZeitAnzeigen()`, die für die laufende Zeitanzeige verantwortlich ist, ruft zunächst die Unterfunktion `ZeitBerechnen()` auf. Diese Funktion erzeugt bei jedem Aufruf ein neues Datumsobjekt mit dem aktuellen Zeitpunkt. Das Ergebnis wird in der Variablen `Immernoch` gespeichert. Zurückgegeben wird jedoch nicht der aktuelle Zeitpunkt, sondern die Differenz zwischen dem aktuellen Zeitpunkt und dem Startzeitpunkt, der beim Einlesen der Datei ermittelt wurde. Der Rückgabewert erfolgt in Sekunden. Da beim Datumsobjekt intern in Millisekunden gerechnet wird, wird der Wert durch 1000 geteilt, um Sekunden zu erhalten. Dabei können jedoch Bruchzahlen zustande kommen. Um eine glatte Sekundenzahl zu erhalten, wendet die aufrufende Funktion `ZeitAnzeigen()` die Methode `Math.round()` an.

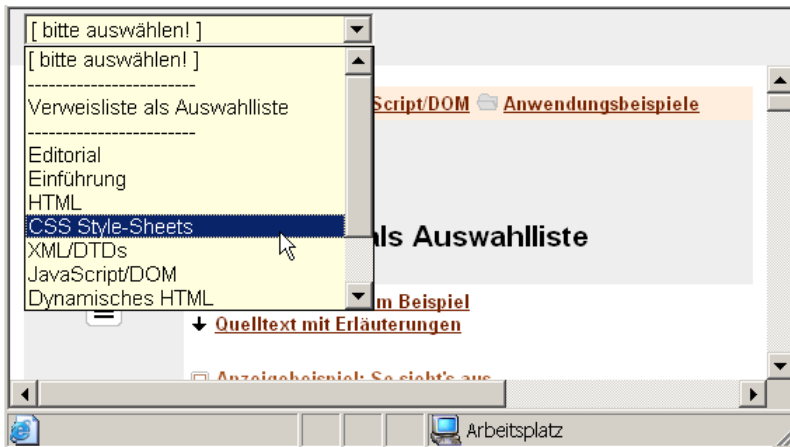
Die absolute Anzahl der vergangenen Sekunden steht dann in der Variablen `absSekunden`. Von dem darin gespeicherten Wert werden alle folgenden Werte abgeleitet. Mit `Math.round((absSekunden-30)/60)` wird ermittelt, wie viele Minuten die Sekundenzahl bereits impliziert. Wenn die Sekundenzahl beispielsweise 200 lautet, sind darin 3 Minuten (180 Sekunden) enthalten. Dieser Wert wird in der Variablen `absMinuten` gespeichert. Da Opera 5.12 in den ersten 30 Sekunden als Ergebnis der Operation `-0` zurückliefert, wird zur Unterdrückung des Minuszeichens mit `Math.abs` der absolute Betrag des Wertes ermittelt. Der Restwert an Sekunden, also bei 200 und 180 etwa der Wert 20, wird mit der Modulo-Division `absSekunden % 60` in der Variablen `relSekunden` gespeichert. Im Falle von 200 steht dann also in `absMinuten` der Wert 3 und in `relSekunden` der Wert 20.

Um die Anzeige der Minuten und Sekunden für die Ausgabe ordentlich zu formatieren, werden Zeichenketten erzeugt, die in den Variablen `anzSekunden` und `anzMinuten` gespeichert werden. Wenn eine Minuten- oder Sekundenzahl einstellig ist, bekommt sie eine 0 vorangestellt.

Mit der Zuweisung an `window.document.Anzeige.Zeit.value` wird die ermittelte und formatierte Zeit anschließend in das Formularfeld geschrieben, das zur Zeitanzeige dient. Damit die Zeitanzeige weiterläuft, ruft sich die Funktion `ZeitAnzeigen()` am Ende mit einer Verzögerungszeit von 1000 Millisekunden, also einer Sekunde, selbst wieder auf. Dadurch wird der gesamte Vorgang wiederholt.

## **19.9 Verweisliste als Auswahlliste**

In diesem Beispiel wird beschrieben, wie Sie eine Auswahlliste eines HTML-Formulars als praktische Alternative für lange Listen mit Verweisen benutzen können. Die Auswahlliste wird dabei in Form einer kleinen Aufklappliste realisiert, die in einem eigenen schmalen Frame-Fenster platziert ist. Das nimmt kaum Platz in Anspruch, und trotzdem steht dem Anwender in jeder Situation eine Liste mit Verweisen zur Verfügung.



19.10: Die Auswahl eines Listenelementes führt einen Sprung zur gewählten Seite aus.



### Quelltext mit Erläuterungen

Das Beispiel zeigt die HTML-Datei, die das Frameset definiert, und die Datei, die im oberen, schmalen Frame-Fenster angezeigt wird. In dieser Datei ist die Auswahlliste und das JavaScript zum Steuern der Auswahlliste enthalten.

Beispiel – Datei mit dem Frameset:

```
<html>
<head>
<title>Verweisliste als Auswahlliste</title>
</head>
<frameset rows="40,*" border="0" frameborder="0" framespacing="0">
  <frame src="verweisliste2.htm" name="oben" marginheight="3" marginwidth="10">
  <frame src="../verweisliste.htm" name="unten">
</frameset>
</html>
```

Beispiel – Datei *verweisliste2.htm*:

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function Go(x) {
  if(x == "nothing") {
    document.forms[0].reset();
    document.forms[0].elements[0].blur();
    return;
  }
  else if(x == "end")
```

```

        top.location.href = parent.frames[1].location;
    else {
        parent.frames[1].location.href = x;
        document.forms[0].reset();
        document.forms[0].elements[0].blur();
    }
}
//-->
</script>
</head>
<body bgcolor="#EEEEEE">
<form action=""><select size=1 name="Auswahl"

onChange="Go(this.form.Auswahl.options[this.form.Auswahl.options.selectedIndex].value)"
    style="width:250px; background-color:#FFFFE0; font-size:9pt; font-
family:Arial,sans-serif;"
    width="250">
<option value="nothing">[ bitte auswählen! ]</option>
<option value="nothing">-----</option>
<option value="../verweisliste.htm">Verweisliste als Auswahlliste</option>
<option value="nothing">-----</option>
<option value="../../editorial/index.htm">Editorial</option>
<option value="../../intro/index.htm">Einführung</option>
<option value="../../html/index.htm">HTML</option>
<option value="../../css/index.htm">CSS Stylesheets</option>
<option value="../../xml/index.htm">XML/DTDs</option>
<option value="../../javascript/index.htm">JavaScript/DOM</option>
<option value="../../dhtml/index.htm">Dynamisches HTML</option>
<option value="../../cgiperl/index.htm">CGI/Perl</option>
<option value="nothing">-----</option>
<option value="end">Beenden
</select></form>
</body>
</html>

```

### Frameset definieren

Da die Auswahlliste im Beispiel während der Anzeige der Web-Seite ständig eingeblendet sein soll, wird ein Frameset definiert. Es besteht aus zwei Fenstern, nämlich einem schmalen Streifen im oberen Bereich des Browser-Fensters und einem Hauptfenster für die Anzeige. Das obere Fenster, in dem die Auswahlliste platziert werden soll, ist 40 Pixel hoch, nimmt also nur wenig Platz in Anspruch.

Im oberen Frame-Fenster wird beim Aufruf die Datei *verweisliste2.htm* angezeigt. Diese Datei wird die Auswahlliste enthalten. Im unteren Fenster wird im Beispiel die Datei *../verweisliste.htm* angezeigt. Das könnte beispielsweise die Titelseite Ihres Web-Projekts sein.

### Auswahlliste definieren

In der Datei *verweisliste2.htm* wird innerhalb des Dateikörpers nichts anderes als ein Formular mit einer Auswahlliste definiert. Die einzelnen Auswahlmöglichkeiten stellen die Verweise dar. Es gibt im Beispiel drei Formen, wie so eine Auswahlmöglichkeit aussieht.

- `<option value="date1.htm">Erste Datei`  
Eine solche Auswahlmöglichkeit definiert Daten, die von JavaScript später als Verweis interpretiert werden. Hinter dem Attribut `value=` steht das gewünschte Verweisziel. Hinter dem `<option>`-Tag steht der Verweistext, der in der Auswahlliste erscheinen soll.
- `<option value="date1.htm">Erste Datei` oder `<option value="nothing">[ Titel ]`  
Mit solchen Einträgen werden Auswahlmöglichkeiten definiert, die in der Liste erscheinen, aber keine Wirkung haben sollen. Das können in längeren Listen zum Beispiel Trennlinien-Einträge sein. Auch die erste Auswahlmöglichkeit der Liste muss so definiert werden, denn sie stellt die »Listenüberschrift« dar. Bei solchen Einträgen notieren Sie einfach hinter dem Attribut `value=` den Wert `nothing`.
- `<option value="end">Beenden`  
Mit einem solchen Eintrag, der sinnvollerweise die letzte Auswahlmöglichkeit darstellt, können Sie bewirken, dass die Auswahlliste beendet wird. Notieren Sie dazu hinter dem Attribut `value=` den Wert `end`. Wenn Sie kein Beenden der Auswahlliste ermöglichen wollen, lassen Sie einen solchen Eintrag einfach weg.

Entscheidend zum Verarbeiten der Auswahlliste sind die Angaben im einleitenden `<select>`-Tag. Dort wird der Event-Handler `onChange=` notiert, der das Ausführen von JavaScript-Code ausführt, sobald der Anwender einen neuen Eintrag in der Auswahlliste ausgewählt. Wenn dieses Ereignis eintritt, wird im Beispiel die Funktion `Go()` aufgerufen. Dazu wird der Wert (`value`) des ausgewählten Listeneintrags als Parameter übergeben. Es handelt sich um den Wert, der entweder ein Verweisziel oder eines der vereinbarten Schlüsselwörter `nothing` oder `end` ist.

### Auswahl verarbeiten

Im Dateikopf steht die Funktion `Go()`, die den ausgewählten Eintrag als Parameter übergeben bekommt und verarbeitet. Die Funktion unterscheidet die drei Fälle `nothing`, `end` oder Verweisziel und führt abhängig davon entsprechende Anweisungen aus.

Wenn ein Eintrag mit dem Wert `nothing` ausgewählt wurde, wird die Auswahlliste zurückgesetzt (`reset()`). Der erste Eintrag, also der Überschrifteneintrag der Liste, wird wieder angezeigt. Der Auswahlbalken wird aus der Liste bis zur nächsten Auswahl entfernt (`blur()`).

Lautet der Eintrag `end`, wird einfach das aktuelle Frame-Fenster, also das mit der Auswahlliste, beendet. Dazu wird der Inhalt des Hauptframe-Fensters zum Inhalt des gesamten Browser-Fensters gemacht (`top.location.href = parent.frames[1].location`).

In allen anderen Fällen wird der übergebene Wert als Verweis interpretiert. Mit der Zuweisung des übergebenen Werts an `parent.frames[1].location.href` wird ein Verweis ausgeführt, der das ausgewählte Verweisziel im Hauptframe-Fenster anzeigt. Auch in diesem Fall wird die Auswahlliste zurückgesetzt (`reset()`). Der erste Eintrag, also der Überschrifteneintrag der Liste,

wird wieder angezeigt. Der Auswahlbalken wird aus der Liste bis zur nächsten Auswahl entfernt (`blur()`).

**Beachten Sie:** Im HTML-Standard ist die Verwendung des Attributes `width=` für das `<select>`-Tag nicht erlaubt. Netscape 4.x interpretiert jedoch keine CSS-Eigenschaften für dieses HTML-Element, akzeptiert dagegen das Attribut `width=` zur Bestimmung der Breite.

Im HTML-Standard ist auch die Verwendung der Attribute `border=`, `frameborder=` und `framespacing=` für das `<frameset>`-Tag nicht erlaubt. Da die Browser jedoch den Standard nicht vollständig interpretieren, wurden diese Attribute im Interesse des Layouts im obigen Beispiel trotzdem verwendet.

## 19.10 Fehlerbehandlung mit `onError`

Der Sprachumfang von JavaScript erlaubt es mittlerweile, umfangreiche Anwendungen zu erstellen. Dynamische Web-Seiten enthalten teilweise umfangreiche Scripts, die in ihren Abläufen auf etliche Variablen zurückgreifen. Dabei können sich leicht unkontrollierbare Werte und Zustände im Programmablauf einschleichen, zum Beispiel auch durch nicht einkalkuliertes Anwenderverhalten. Seit der JavaScript-Version 1.1 steht Ihnen mit dem Event-Handler `onError` ein Werkzeug zur Verfügung, um innerhalb des Scripts Fehler abzufangen und gezielt darauf zu reagieren.

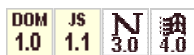
Zur Unterdrückung von Fehlermeldungen, die zur Laufzeit eines Scripts entstehen, notieren Sie im Kopf der Datei `window.onerror=`. Als Wert weisen Sie den Namen einer Funktion zu, welche die Fehlerbehandlung ausführt. Der Rückgabewert der Funktion muss `true` sein. Damit haben Sie Ihre eigene Fehlerbehandlung.

Der Fehlerbehandlungs-Funktion können Sie bis zu 3 optionale Parameter übergeben:

1. **Nachricht:** enthält die Fehlerbeschreibung des Fehlers
2. **Datei:** enthält den URI der fehlerverursachenden Datei
3. **Zeile:** enthält die Zeile, in der der Fehler auftritt

Diese Art der Fehlerkontrolle können Sie für Netscape 4.x und MS Internet Explorer ab Version 4.x verwenden. Netscape 6 ist leider nicht in der Lage, die Parameter vollständig zu interpretieren. Er speichert jeweils im ersten Parameter das auslösende Event-Objekt. Opera 5.12 interpretiert den an das Fensterobjekt gebundenen Event-Handler `window.onError` nicht.

Berücksichtigen Sie, dass Sie nur logische Fehler überwachen und unterdrücken können, die während der Laufzeit der Scripts entstehen. Syntaxfehler, wie z.B. fehlende Klammern usw., sind damit nicht abfangbar.



### Quelltext mit Erläuterungen

Das Beispiel zeigt eine einfache Variante der Fehlerbehandlung. Am Ende des JavaScript-Blocks wird eine Funktion aufgerufen, die nicht existiert.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
window.onerror = meinFehler;

function meinFehler(Nachricht,Datei,Zeile)
{
  Fehler = "Fehlermeldung:\n"+ Nachricht+"\n"+Datei+"\n"+Zeile;
  zeigeFehler();
  return true;
}

function zeigeFehler()
{
  alert(window.Fehler)
}
nichtda();

//-->
</script>
</head><body>
<a href="javascript:zeigeFehler()">Fehler</a><br>
</body></html>
```

Beim Einlesen der Datei wird mit `window.onerror = meinFehler` angewiesen, im Fehlerfall die Funktion `meinFehler()` aufzurufen. Dieses Ereignis tritt im Beispiel am Ende des Script-Bereichs ein, wo versucht wird, die nichtvorhandene Funktion `nichtda()` aufzurufen.

### **Die Funktion `meinFehler()`**

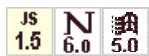
Die Funktion speichert mit Zeichenkettenverknüpfung die automatisch übergebenen Parameter in der Variablen `Fehler`. Dann ruft sie im Beispiel die Funktion `zeigeFehler()` auf. Diese Funktion dient im Beispiel lediglich dazu, den Fehler zu Demonstrationszwecken auszugeben. Mit `return true` wird erreicht, dass der Fehler im Browser nicht angezeigt wird. Mit dem Rückgabewert `false` würden Sie bewirken, dass der Browser den Fehler anmeckert.

In einer komplexeren Anwendung hätte die Funktion `meinFehler()` also alle Möglichkeiten, auf den Fehler zu reagieren. Sie könnte beispielsweise die Zeichenkette des übergebenen Parameters `Nachricht` untersuchen, um herauszufinden, welcher Fehler genau aufgetreten ist. Je nach Situation könnte eine solche Funktion bewirken, dass das Script zu einem früheren Punkt im Programmablauf zurückspringt, Variablen zurücksetzt, den Cursor in ein bestimmtes Formularfeld setzt usw.



## 19.11 Fehlerbehandlung mit dem Statement try..catch

Mit Hilfe der Fehlerbehandlung mit dem Event-Handler `onError` können Sie in einem JavaScript Fehler abfangen, **nachdem** diese aufgetreten sind. Mit der hier vorgestellten Methode können Sie jedoch bereits im Vorfeld verhindern, dass in kritischen Situationen überhaupt Fehler auftreten. Dazu steht seit der JavaScript-Version 1.5 das `try..catch`-Statement zur Verfügung. Es erlaubt Ihnen, Variablen und Werte zu überprüfen und je nach Ergebnis zu reagieren. Dieses Statement wird vom MS Internet Explorer ab Version 5.0, von Netscape ab Version 6.0 und von Opera ab Version 5 interpretiert.



### Quelltext mit Erläuterungen

Das Beispiel ist so konstruiert, dass es auf eine Variable zugreift, die zunächst noch gar nicht existiert. Das würde zu einem Fehler führen. Das Script »weiß« jedoch, dass dieser Fehler auftreten kann, und überprüft deshalb mit dem `try..catch`-Statement laufend, ob die Variable schon existiert. Erst wenn die Variable existiert, wird die Überwachung beendet.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--

setTimeout("x=3",200);

function zeigeErgebnis(Zaehler,Ergebnis)
{
    alert("Nach "+(Zaehler)+ " Durchläufen existierte x.\nDie Zahl x ist
"+Ergebnis+".");
}

function teste_x(Zaehler)
{
    try {
        if (x==2) throw "richtig";
        else if (x==3) throw "falsch";
    }
    catch (e) {
        if (e=="richtig") {zeigeErgebnis(Zaehler,e);return;}
        else if (e=="falsch") {zeigeErgebnis(Zaehler,e);return;}
    }
    finally { Zaehler++; }
}

setTimeout('teste_x('+Zaehler+')',30);
}
teste_x(0)
//-->
</script>
</head><body>
</body></html>
```

Im Dateikopf der Datei ist ein Script-Bereich notiert. Darin werden die erste und die letzte Anweisung sofort beim Einlesen ausgeführt. Die restlichen Anweisungen stehen in Funktionen.

### **Die Variable `x`**

Während des Ladens der Datei wird eine Variable `x` zeitverzögert mit dem Wert 3 belegt. Dazu dient die Methode `setTimeout()`. Die Variable ist also erst nach 200 Millisekunden verfügbar. Jeder Versuch, vorher auf diese Variable zuzugreifen, würde zu einem Fehler führen. Die nächste Anweisung, die im Beispiel direkt ausgeführt wird, ist die letzte im Script-Bereich, nämlich `teste_x(0)`. Damit wird die Funktion `teste_x()` aufgerufen, die oberhalb notiert ist.

### **Die Funktion `teste_x()`**

Diese Funktion versucht, auf die Variable `x` zuzugreifen. Da zum Zeitpunkt des Aufrufes der Funktion jedoch noch nicht sicher ist, ob die Variable `x` bereits existiert, ist innerhalb der Funktion zur Vermeidung von Fehlermeldungen das `try..catch`-Statement notiert. Beim Aufruf erhält die Funktion `teste_x()` einen Parameter namens `Zaehler` übergeben. Das dient im Beispiel zu Kontrollzwecken.

### **Aufbau des `try..catch`-Statements**

Das `try..catch`-Statement hat generell folgenden Aufbau: Nach dem Schlüsselwort `try` (*try* = *versuchen*) wird eine öffnende geschweifte Klammer, gefolgt von der zu prüfenden Bedingung, notiert. Je nach Erfordernissen können Sie dann mit `throw` (*throw* = *auswerfen*) eigene Fehler definieren. Die `throw`-Definition ist jedoch optional. Anschließend folgt eine schließende geschweifte Klammer, die den `try`-Block beendet.

Daran anschließend notieren Sie das Schlüsselwort `catch` (*catch* = *abfangen*). `catch` hat Funktions-Charakter und erwartet einen Parameter `e`. Die Variable `e` ist erforderlich, da Sie über diese Variable letztlich die Reaktion des Scripts auf den aktuellen Zustand kontrollieren. Den Namen der Variablen können Sie frei wählen (der Name muss also nicht `e` sein). Innerhalb des Funktionsblocks von `catch()`, der wie üblich in geschweiften Klammern steht, können Sie die mit `throw` definierten Fehler auswerten und darauf reagieren.

Zuletzt können Sie noch das Schlüsselwort `finally` notieren. In dem davon abhängigen Anweisungsblock können Sie weitere Anweisungen notieren. Diese Anweisungen werden unabhängig von der Fehlerbehandlung in jedem Fall ausgeführt.

### **Anwendung des `try..catch`-Statements im Beispiel**

Im ersten Teil der Anweisung wird geprüft, ob die Variable `x` den Wert 2 oder 3 besitzt. Je nach Ergebnis werden mit `throw` verschiedene Fehlerwerte definiert. Hat `x` den Wert 2, so wird der »Fehler« mit dem Wert `richtig` generiert. Hat sie den Wert 3, so erhält der Fehler den Wert `falsch`. Weitere Fehlervarianten werden nicht behandelt.

Im ersten Durchlauf existiert die Variable `x` im Beispiel noch gar nicht, da sie ja erst nach 200 Millisekunden auftaucht. Sie kann damit weder den Wert 2 noch den Wert 3 besitzen. In der nachfolgenden Fehlerbehandlungsroutine `catch(e)` wird geprüft, ob einer der definierten Fehler, also `richtig` oder `falsch`, aufgetreten ist. Zunächst ist das offensichtlich nicht der Fall. Die

Anweisungen, die von den »Fehlerwerten« `richtig` und `falsch` abhängig sind, werden deshalb nicht ausgeführt. Die `finally`-Anweisung wird dagegen in jedem Fall ausgeführt. Sie bewirkt im Beispiel, dass der übergebene Parameter `Zaehler` um 1 erhöht wird.

### **Gesamtkontrolle**

Am Ende ruft sich die Funktion `teste_x()` mit `setTimeout()` um 30 Millisekunden zeitverzögert selbst wieder auf. So behält sie die Kontrolle über das Geschehen, bis ein definierter Zustand eintritt. Der Parameter `Zaehler` wird dabei mit Hilfe einer Zeichenkettenverknüpfung übergeben.

Interessant wird es, wenn der Zeitpunkt erreicht ist, zu dem die Variable `x` existiert. In diesem Fall tritt einer der vordefinierten Fälle ein. Da `x` den Wert 3 besitzt, wird der `throw`-Fehler mit dem Wert `falsch` generiert (dies soll im Beispiel einfach zeigen, dass `throw` zur Erzeugung von Werten gedacht ist, die durchaus und oft auch Fehlerzustände bezeichnen). Im nachfolgenden `catch(e)`-Block führt dies dazu, dass die Funktion `zeigeErgebnis()` aufgerufen wird. Im Beispiel wird für beide definierten `throw`-Werte die gleiche Funktion aufgerufen. Sie können an dieser Stelle jedoch auch völlig verschiedene Anweisungen notieren. Jede dieser Fehlerbehandlungsroutinen bricht gleichzeitig die Funktion `teste_x()` mit `return ab`, da `x` ja nun existiert und der »kritische Zustand« beendet ist.

### **Ausgabe des Ergebnisses**

Die Funktion `zeigeErgebnis()` erhält als Parameter die Variablen `Zaehler` und `Ergebnis` übergeben. In der Variablen `Zaehler` ist die Anzahl der Durchläufe bis zur Existenz der Variablen `x` gespeichert und in der Variablen `Ergebnis` das Resultat der Fehlerbehandlung. Mit `alert()` wird im Beispiel zur Kontrolle ausgegeben, wie viele Durchläufe benötigt wurden und was für ein Ergebnis erreicht wurde.

Die Ausgabe zeigt übrigens sehr deutlich, wie unterschiedlich die Browser mit `setTimeout()` umgehen. Während der MS Internet Explorer bei Tests im Schnitt fünf Durchläufe benötigte, brauchte Netscape 6.x im Schnitt zwei Durchläufe und Opera vier.

### **Anwendungsfälle**

Prüfungen mit dem `try...catch`-Statement sind z.B. dann sinnvoll, wenn Sie wie im Beispiel mit `setTimeout()` zeitversetzte Aktionen ausführen und davon abhängige Anweisungen ausführen wollen. Ebenfalls sinnvoll ist das Statement, wenn Sie z.B. auf Variablen oder Funktionen zugreifen wollen, die in anderen Frame-Fenstern notiert sind, wobei das Script nicht wissen kann, ob die Datei im anderen Frame-Fenster, in der das entsprechende Script notiert ist, bereits eingelesen oder überhaupt die dort aktuell angezeigte Seite ist.

## **19.12 Änderungen der Fenstergröße überwachen**

Ein Verändern der Fenstergröße durch den Anwender hat bei Netscape 4.x gelegentlich die Folge, dass dieser gesetzte Variablen, dynamische Positionierungen oder Stylesheet-Angaben »vergisst«. Es ist jedoch möglich, Änderungen der Fenstergröße zu überwachen und damit gezielt

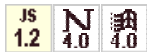
auf die Änderung zu reagieren. Auch beim MS Internet Explorer oder bei Netscape 6.x ist es gelegentlich sinnvoll, auf Änderungen reagieren zu können.

Für die Überwachung der Fenstergröße wird der Event-Handler `onResize` verwendet. Dieser Event-Handler tritt ein, wenn der Anwender die Fenstergröße ändert. Sie können diesem Event-Handler eine Funktion zuordnen, die bei Größenänderungen ausgeführt wird.

Beachten Sie, dass Sie mit diesem Beispiel nur den Neuaufbau der Seite erreichen. Eventuell geänderte Variablen und Werte gehen dabei verloren. In Opera 5.12 ist es **nicht** möglich, die Änderung der Fenstergröße mit dem Event-Handler `onResize` zu überwachen.



19.11: Das Beispielscript gibt die aktuelle Fenstergröße bei jeder Größenänderung neu aus.



### Quelltext mit Erläuterungen

Im Beispiel wird die Fenstergröße auf Änderungen hin überwacht. Sobald sie geändert wird, erfolgt ein Neuaufbau der Seite und die geänderten Größen werden jeweils mit `document.write()` zur Kontrolle in das Dokument geschrieben.

```
<html><head>
<title>Größenänderung überwachen</title>
<script type="text/javascript">
<!--

function Fensterweite()
{
  if (window.innerWidth) return window.innerWidth;
  else if (document.body && document.body.offsetWidth) return
document.body.offsetWidth;
  else return 0;
}

function Fensterhoehe()
{
  if (window.innerHeight) return window.innerHeight;
  else if (document.body && document.body.offsetHeight) return
document.body.offsetHeight;
  else return 0;
}
```

```

function neuAufbau()
{
    if (Weite != Fensterweite() || Hoehe != Fensterhoehe())
        window.history.go(0);
}

/*Überwachung von Netscape initialisieren*/
if(!window.Weite && window.innerWidth)
{
    window.onresize = neuAufbau;
    Weite = Fensterweite();
    Hoehe = Fensterhoehe();
}

//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
/*Überwachung von MS Internet Explorer initialisieren*/
if(!window.Weite && document.body && document.body.offsetWidth)
{
    window.onresize = neuAufbau;
    Weite = Fensterweite();
    Hoehe = Fensterhoehe();
}
//-->
</script>
<div id="Beispiel" style="position:absolute;top:100px;left:100px;border:solid 1px
#000000;">
Text
</div>
<script type="text/javascript">
<!--
    document.write("Weite: "+Weite +" Höhe: "+Hoehe);
//-->
</script>
</body></html>

```

### Die Funktionen *Fensterweite()* und *Fensterhoehe()*

Diese beiden Funktionen dienen dazu, die Fenstergröße zu ermitteln. Die Funktion `Fensterweite()` prüft, ob der Browser die Eigenschaft `window.innerWidth` kennt (das ist bei Netscape der Fall, nicht aber beim Internet Explorer). Ist dies der Fall, so wird der Wert, der in dieser Eigenschaft gespeichert ist, zurückgegeben. Kennt der Browser diese Eigenschaft nicht, so wird geprüft, ob er das Objekt `document.body` kennt und ob dieses Objekt die Eigenschaft `document.body.offsetWidth` kennt (das ist beim Internet Explorer der Fall, nicht aber bei Net-

scape). Tritt dieser Zustand ein, so wird der Wert der Eigenschaft `document.body.offsetWidth` zurückgegeben. Sollte der Browser auch diesen Zweig der Anweisung nicht verfolgen können, wird 0 zurückgegeben.

Die Funktion `Fensterhoehe()` arbeitet analog und verwendet die Eigenschaften `window.innerHeight` bzw. `document.body.offsetHeight`.

### **Überwachung für Netscape initialisieren**

Beim Laden der Datei existiert die Variable `Weite` noch nicht. Da eine globale Variable immer eine Eigenschaft bzw. ein Objekt des Fensterobjektes ist, kann mit `window.Weite` geprüft werden, ob diese existiert. Netscape kennt zudem die Eigenschaft `window.innerWidth`. Deshalb arbeitet er die entsprechende Anweisung ab. Zuerst wird die Überwachung des Event-Handler `onResize` initialisiert (zum Verständnis siehe auch die allgemeine Verwendung des event-Objekts). `onResize` wird ausgelöst, sobald sich die Fenstergröße ändert, und ruft dann die Handler-Funktion `neuAufbau()` auf. Anschließend wird in der Variablen `Weite` der Rückgabewert der Funktion `Fensterweite()` und in der Variablen `Hoehe` der Rückgabewert der Funktion `Fensterhoehe()` gespeichert. Damit stehen beide Variablen für weitere Verarbeitungsaufgaben zur Verfügung.

### **Überwachung für den MS Internet Explorer initialisieren**

Die Überwachung für den Internet Explorer gestaltet sich ähnlich. Im Unterschied zum Netscape Navigator benötigt er jedoch das Objekt `document.body`. Dieses Objekt steht jedoch erst nach dem Laden des Body-Tags zur Verfügung. Deshalb ist der entsprechende Scriptbereich nicht im Dateikopf, sondern im `body`-Bereich des Dokuments notiert. Auch Netscape 6 kennt das Objekt `document.body` und dessen Eigenschaften. Er führt jedoch diesen Scriptbereich nicht aus, da dieser Browser zum Laufzeitpunkt die Variable `Weite` bereits kennt.

### **Die Handler-Funktion `neuAufbau()`**

Die Funktion `neuAufbau()` wird aufgerufen, wenn der Event-Handler `onResize` aktiv wird. Die Funktion vergleicht den Wert der Variablen `window.Weite` mit dem Rückgabewert der Funktion `Fensterweite()` und den Wert der Variablen `window.Hoehe` mit dem Rückgabewert der Funktion `Fensterhoehe()`. Daran erkennt sie, ob sich die Fenstergröße verändert hat. In diesem Fall wird durch die Verwendung der Methode `history.go()` ein Neuaufbau der Seite veranlasst, ohne dass diese vom Server erneut geladen werden muss. Wird der Event-Handler `onResize` aufgerufen, ohne dass eine reale Veränderung der Fenstergröße stattgefunden hat – das kann gelegentlich passieren –, geschieht nichts, da die Bedingung zum Neuladen nicht erfüllt ist.

## 20 JavaScript: Objekt-Referenz

Dieses Kapitel stellt die vordefinierten Objekte von JavaScript sowie ihre Eigenschaften und Methoden vor. Die Nummerierung vor den Verweisen symbolisiert die JavaScript-Objekt-Hierarchie und hat sonst keine weitere Bedeutung.

|           |                            |                                              |
|-----------|----------------------------|----------------------------------------------|
| 1         | window                     | Anzeigefenster                               |
| 1.1       | frames                     | Frame-Fenster                                |
| 1.2       | document                   | Dokument im Anzeigefenster                   |
| 1.2.1     | <b>HTML-Elementobjekte</b> | alle HTML-Elemente des Dokuments             |
| 1.2.2     | node                       | alle Knoten des Elementbaums                 |
| 1.2.3     | all                        | alle HTML-Elemente des Dokuments – Microsoft |
| 1.2.3.1   | style                      | CSS-Attribute von HTML-Elementen             |
| 1.2.4     | anchors                    | Verweisanker im Dokument                     |
| 1.2.5     | applets                    | Java-Applets im Dokument                     |
| 1.2.6     | forms                      | Formulare im Dokument                        |
| 1.2.6.1   | elements                   | Formularelemente eines Formulars             |
| 1.2.6.1.1 | options                    | Optionen einer Auswahlliste eines Formulars  |
| 1.2.7     | images                     | Grafikreferenzen im Dokument                 |
| 1.2.8     | embeds                     | Multimedia-Referenzen im Dokument            |
| 1.2.9     | layers                     | Layer im Dokument – Netscape                 |
| 1.2.10    | links                      | Verweise im Dokument                         |
| 1.3       | event                      | Anwenderereignisse                           |
| 1.4       | history                    | besuchte Seiten                              |
| 1.5       | location                   | URIs                                         |
| 2         | Array                      | Ketten von Variablen                         |
| 3         | Boolean                    | Ja-Nein-Variablen                            |
| 4         | Date                       | Datum und Uhrzeit                            |
| 5         | Function                   | JavaScript-Funktionen                        |
| 6         | Math                       | Berechnungen                                 |
| 7         | navigator                  | Browser-Informationen                        |
| 7.1       | mimeType                   | MimeType-Informationen                       |
| 7.2       | plugins                    | installierte Plug-Ins                        |
| 8         | Number                     | numerische Werte                             |
| 9         | RegExp                     | reguläre Ausdrücke                           |
| 10        | Screen                     | Bildschirminformationen                      |
| 11        | string                     | Zeichenketten                                |

## 20.1 Hinweise zur Objekt-Referenz

### 20.1.1 Objekthierarchie

Es gibt zwei Gruppen von vordefinierten Objekten im klassischen JavaScript.

- Die eine Gruppe ist diejenige, die das Browser-Fenster – markiert durch das `window`-Objekt – zum Ausgangspunkt hat. Das Browser-Fenster ist der allgemeine Rahmen für alles, was innerhalb davon angezeigt wird. Das, was innerhalb davon angezeigt wird, gilt als Dokument, markiert durch das `document`-Objekt. Letzteres ist wiederum für das Document Object Model (DOM) das oberste Ausgangsobjekt. Und ausgehend vom `document`-Objekt gibt es leider zwei verschiedene »Objekt-Philosophien«, die heute nebeneinander stehen: Im klassischen JavaScript enthält ein Dokument nämlich bestimmte Elemente, auf die mit JavaScript zugegriffen werden kann, zum Beispiel Grafiken, markiert durch das `images`-Objekt, oder Formulare, markiert durch das `forms`-Objekt. Formulare wiederum bestehen aus Formularelementen – markiert durch das `elements`-Objekt. Im neueren Modell des Internet Explorers und auch des DOM ist dagegen nicht nur der Zugriff auf solche bestimmten Elementtypen möglich, sondern der Zugriff auf **alle Elemente** eines Dokuments – nach Syntax des Internet Explorers markiert durch das `all`-Objekt und nach DOM-Syntax markiert durch bestimmte Zugriffsmethoden des `document`-Objekts, durch die HTML-Elementobjekte und das `node`-Objekt. Diese beiden widerstreitenden Modelle zwischen klassischem JavaScript und den neueren Modellen sorgen leider für viel Verwirrung. Die neueren Modelle sind zweifellos durchdachter und bieten dem JavaScript-Programmierer viel mehr Möglichkeiten. Doch die »Jugendsünden« von Netscapes klassischem JavaScript sind nun mal nicht so leicht aus der Welt zu schaffen, und so bestehen die Modelle auf unabsehbare Zeit weiter nebeneinander.
- Die zweite Gruppe von JavaScript-Objekten haben nichts direkt mit dem Geschehen im Anzeigefenster zu tun, liefern aber wichtige andere Daten oder ausüben Funktionen. Netscape bezeichnet sie als **core objects** (*Kernobjekte*). Typische Vertreter dieser zweiten Gruppe sind etwa das `Date`-Objekt für Datums- und Uhrzeitberechnungen, das `navigator`-Objekt für Informationen zum verwendeten Browser, oder das `Array`-Objekt zum Erzeugen von Serienvariablen. Auch bei dieser Gruppe gibt es ein paar hierarchische Beziehungen. So ist beispielsweise das `plugins`-Objekt ein Unterobjekt des `navigator`-Objekts.

Objekthierarchien werden in JavaScript stets dadurch ausgedrückt, dass die Objekte aneinandergereiht und durch Punkt getrennt werden. Typische Notationen sehen so aus wie `window.document.images.length`. Dabei steht `window` für das Browser-Fenster, in dem das JavaScript läuft, `document` für das in diesem Fenster angezeigte Dokument, `images` für die Gesamtheit der in dem Dokument enthaltenen Grafikreferenzen, und `length` für eine Eigenschaft des `images`-Objekts.

Abgesehen von dieser einfachen Grundregel haben viele Objekte jedoch ihre Besonderheiten. Bei den Beschreibungen zu den jeweiligen Objekten wird darauf hingewiesen.

### 20.1.2 Objekt-Referenz verwenden

Jedes JavaScript-Objekt wird in dieser Referenz nach folgendem Schema beschrieben:



## Allgemeines zur Verwendung

In diesem Abschnitt wird beschrieben, welche Voraussetzungen erfüllt sein müssen, damit Sie die Eigenschaften und Methoden des Objekts benutzen können. Lesen Sie diesen Abschnitt genau, wenn Sie mit dem Objekt noch nie gearbeitet haben.

## Eigenschaften

In diesem Abschnitt werden die Eigenschaften eines Objekts beschrieben. Eine Eigenschaft eines JavaScript-Objekts ist zum Beispiel `window.location.href` – der URI der aktuell angezeigten HTML-Datei. Die Werte von solchen Objekteigenschaften können Sie in jedem Fall auslesen. In vielen Fällen können Sie die Werte auch ändern. In der Objekt-Referenz wird jeweils angegeben, ob Sie eine gespeicherte Objekteigenschaft nur auslesen oder auch ändern können. Dazu werden in dieser Objekt-Referenz folgende Signalisierungen benutzt:

| Symbol | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | Sie können den Wert der Eigenschaft auslesen, aber nicht ändern. Sie können den Wert zum Beispiel in einer selbst definierten Variablen speichern. So können Sie beispielsweise den URI der angezeigten HTML-Datei durch <code>var Adresse = window.location.href</code> in einer Variablen namens <code>Adresse</code> speichern. Das heißt, Sie weisen einer Variablen den Wert der Objekteigenschaft zu. Die Variable können Sie nach Belieben weiterverarbeiten. |
|        | Sie können die Objekteigenschaft auslesen, ihr aber auch selbst einen neuen Wert zuweisen. So können Sie zum Beispiel den URI der angezeigten HTML-Datei durch <code>window.location.href = "http://selfhtml.teamone.de/"</code> ändern. Das heißt, Sie weisen der Objekteigenschaft einen Wert zu. Im Beispiel hat das zur Folge, dass der Web-Browser einen Sprung zu dem angegebenen URI ausführt.                                                                |

## Methoden

Methoden eines Objekts sind objektgebundene Funktionen, die eine Aktion ausführen. So können Sie zum Beispiel mit `document.write("<b>Ihr Web-Browser: </b>" + navigator.userAgent)` dynamisch erzeugten, HTML-formatierten Text in das aktuelle Anzeigefenster des Web-Browsers schreiben. Im Beispiel wird auf diese Weise der vom Anwender verwendete Web-Browser-Typ ins Anzeigefenster geschrieben.

## 20.2 window

### Eigenschaften

|                            |                                    |
|----------------------------|------------------------------------|
| <code>closed</code>        | (geschlossenes Fenster)            |
| <code>defaultStatus</code> | (Normalanzeige in der Statuszeile) |
| <code>innerHeight</code>   | (Höhe des Anzeigebereichs)         |
| <code>innerWidth</code>    | (Breite des Anzeigebereichs)       |
| <code>locationbar</code>   | (Adresszeile)                      |
| <code>menubar</code>       | (Menüleiste)                       |

|             |                                  |
|-------------|----------------------------------|
| name        | (Fenstername)                    |
| outerHeight | (Höhe des gesamten Fensters)     |
| outerWidth  | (Breite des gesamten Fensters)   |
| pageXOffset | (Fensterstartposition von links) |
| pageYOffset | (Fensterstartposition von oben)  |
| personalbar | (Zeile für Lieblingsadressen)    |
| scrollbars  | (Scroll-Leisten)                 |
| statusbar   | (Statuszeile)                    |
| status      | (Inhalt der Statuszeile)         |
| toolbar     | (Werkzeuggestreife)              |

### Methoden

|                          |                                         |
|--------------------------|-----------------------------------------|
| alert()                  | (Dialogfenster mit Infos)               |
| back()                   | (zurück in History)                     |
| blur()                   | (Fenster verlassen)                     |
| captureEvents()          | (Ereignisse überwachen)                 |
| clearInterval()          | (zeitliche Anweisungsfolge abbrechen)   |
| clearTimeout()           | (Timeout abbrechen)                     |
| close()                  | (Fenster schließen)                     |
| confirm()                | (Dialogfenster zum Bestätigen)          |
| disableExternalCapture() | (Fremdüberwachung verhindern)           |
| enableExternalCapture()  | (Fremdüberwachung erlauben)             |
| find()                   | (Text suchen)                           |
| focus()                  | (Fenster aktiv machen)                  |
| forward()                | (vorwärts in History)                   |
| handleEvent()            | (Ereignis übergeben)                    |
| home()                   | (Startseite aufrufen)                   |
| moveBy()                 | (Bewegen mit relativen Angaben)         |
| moveTo()                 | (Bewegen mit absoluten Angaben)         |
| open()                   | (neues Fenster öffnen)                  |
| print()                  | (ausdrucken)                            |
| prompt()                 | (Dialogfenster für Werteingabe)         |
| releaseEvents()          | (Ereignisse abschließen)                |
| resizeBy()               | (Größe verändern mit relativen Angaben) |
| resizeTo()               | (Größe verändern mit absoluten Angaben) |
| routeEvent()             | (Event-Handler-Hierarchie durchlaufen)  |
| scrollBy()               | (Scrollen um Anzahl Pixel)              |
| scrollTo()               | (Scrollen zu Position)                  |
| setInterval()            | (zeitliche Anweisungsfolge setzen)      |
| setTimeout()             | (Timeout setzen)                        |
| stop()                   | (abbrechen)                             |

### Unterobjekte

document  
event  
history  
location

### window: Allgemeines zur Verwendung

Das Objekt `window` (*Fenster*) ist das oberste Objekt der Objektfamilie für alles, was im Browser-Fenster angezeigt wird. Über das Objekt `window` können Sie das Dokumentfenster abfragen und kontrollieren. Ferner können Sie neue Fenster definieren. Dabei können Sie auch die Fenstereigenschaften frei bestimmen.

Das aktuelle Fenster des Web-Browsers können Sie über die Objekte `window` oder `self` ansprechen. Auf die Objekte `window` und `self` lassen sich alle Eigenschaften und Methoden des `window`-Objekts anwenden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
window.defaultStatus = "Meine Homepage";
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel belegt die ständige Anzeige der Statuszeile des aktuellen Anzeigefensters mit dem Wert "Meine Homepage". Dazu wird die Fenstereigenschaft `defaultStatus` aufgerufen. `window` oder `self` können auch weggelassen werden. So hat `window.moveTo(200,100)` die gleiche Wirkung wie `moveTo(200,100)`. Der Übersichtlichkeit halber ist es jedoch besser, den vordefinierten Objektnamen mit anzugeben. Auch Frame-Fenster innerhalb eines Framesets stellen aus Sicht von JavaScript eigene Fenster dar. Lesen Sie dazu jedoch den Abschnitt über das Objekt `frames`. Sie können mit Hilfe von JavaScript nicht nur vorhandene Fenster ansprechen, sondern auch neue Fenster erzeugen und diese mit Hilfe von Variablen ansprechen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function NeuFenster() {
  MeinFenster =
  window.open("datei2.htm", "Zweitfenster", "width=300,height=200,scrollbars");
  MeinFenster.focus();
}
//-->
</script>
</head><body>
<a href="javascript:NeuFenster()">Neues Fenster</a>
</body></html>
```

Die Beispieldatei enthält einen Verweis. Beim Anklicken dieses Verweises wird die JavaScript-Funktion `NeuFenster()` aufgerufen. Diese Funktion öffnet ein neues, kleines Browser-Fenster und zeigt darin die Datei `datei2.htm` an. Einzelheiten zum Öffnen neuer Fenster werden bei der Methode `open()` beschrieben. Ein solches Fenster ist nach dem Erzeugen mit der Variablen ansprechbar, in der die Fensterinstanz gespeichert ist. Im Beispiel ist das die Variable `MeinFenster`. Über diesen Instanznamen können Sie alle Eigenschaften und Methoden des erzeugten Fensters ansprechen. Ein Beispiel:

```
<a href="javascript:MeinFenster.close()">Fenster zumachen</a>
```

Im Beispiel wird das zuvor geöffnete Fenster beim Klicken auf den Verweis wieder geschlossen. Zuständig dafür ist die Methode `close()`. Das Fenster wird dabei mit seinem Instanznamen angesprochen, im Beispiel also mit dem Namen der Variablen `MeinFenster`, mit der es zuvor erzeugt wurde.

Bei der Variablen `MeinFenster` handelt es sich um eine globale Variable. Diese wurde erzeugt durch eine Deklaration der Variablen ohne das Schlüsselwort `var`. Die Deklaration als globale Variable ermöglicht es Ihnen, dieses Fenster auch außerhalb der erzeugenden Funktion anzusprechen.

Genau so, wie Sie vom Hauptfenster auf ein mit `open()` erzeugtes Zweitfenster zugreifen können, können Sie von einem solchen Zweitfenster auf das Hauptfenster zugreifen. Dazu gibt es das Fenster-Objekt `opener`. Damit wird das Fenster angesprochen, von dem aus das aktuelle Fenster geöffnet wurde. Über das Objekt `opener` können Sie alle Eigenschaften und Methoden des öffnenden Fensters ansprechen.

Testbeispiel (im Zweitfenster):

```
<a href="javascript:opener.close()">Hauptfenster zumachen</a>
```

Notieren Sie nach `opener` einen Punkt und danach die gewünschte Methode oder Eigenschaft.



Speichert die Information, ob ein Fenster, das zum Beispiel mit `open()` neu geöffnet wurde, mittlerweile geschlossen wurde. Hat den Wert `true`, wenn das Fenster geschlossen wurde. Sinnvoll, um die Existenz eines Fensters abzufragen, bevor Anweisungen zum Verändern eines Fensters folgen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var InfoWin = window.open("datei1.htm", "Zweitfenster");
function CheckOpen() {
  if(InfoWin.closed == true) alert("Fenster wurde geschlossen");
  else alert("Fenster noch offen");
}
//-->
</script>
</head><body>
<a href="javascript:CheckOpen()">Fenster zu?</a>
</body></html>
```

Das Beispiel lädt beim Einlesen ein Zweitfenster mit einer anderen Datei `datei1.htm`. Beim Klicken auf den Verweis wird eine Funktion `CheckOpen()` aufgerufen, die ausgibt, ob das andere Fenster noch offen ist oder nicht.



Speichert den Inhalt der Statuszeile, der so lange angezeigt wird, wie kein besonderes Ereignis eintritt (etwa das Überfahren eines Verweises mit der Maus). Das Beispiel belegt beim Einlesen der Datei die normale Anzeige der Statuszeile mit dem Wert `Helmut's Homepage`:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
window.defaultStatus = "Helmut's Homepage";
//-->
</script>
</head><body>
<p>Achten Sie auf die Statuszeile</p>
</body></html>
```

### JS 1.2 N 4.0 Lesen Ändern *innerHeight*

Speichert die Höhe des Anzeigebereichs eines Fensters. Sehr gut geeignet in Verbindung mit absolutem Positionieren von Elementen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
window.innerHeight = 300;
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel setzt beim Einlesen der Datei den Anzeigebereich des Fensters auf eine Höhe von 300 Pixel fest. Beachten Sie beim Neufestsetzen der Fensterhöhe die Bildschirmgröße. Diese können Sie mit dem `Screen`-Objekt ermitteln. Beachten Sie auch, dass der Anzeigebereich eines Fensters um einiges kleiner sein muss als das Fenster selbst, da das Fenster in der Regel noch Elemente wie Titelleiste, Menüleiste usw. enthält. Die absolute Höhe eines Fensters können Sie mit `outerHeight` auslesen bzw. setzen.

**Beachten Sie:** Unter Opera 5.12 können Sie die Eigenschaft `innerHeight` nur auslesen, jedoch nicht setzen. Im MS Internet Explorer können Sie mit `document.body.offsetHeight` die innere Fensterhöhe annähernd bestimmen. Zu diesem Zeitpunkt muss jedoch das `Body`-Element bereits vorhanden sein, d.h. es ist sinnvoll, die Ermittlung dieser Eigenschaft vom Event-Handler `onLoad` abhängig zu machen.

### JS 1.2 N 4.0 Lesen Ändern *innerWidth*

Speichert die Breite des Anzeigebereichs eines Fensters. Sehr gut geeignet in Verbindung mit absolutem Positionieren von Elementen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function BreiteFestlegen() {
  window.innerWidth = document.Eingabe.Feld.value;
}
//-->
</script>
</head><body>
```

```
<form name="Eingabe" action="">
<input type="text" name="Feld">
<input type="button" value="Test" onClick="BreiteFestlegen()">
</form>
</body></html>
```

Das Beispiel erlaubt es, in einem Eingabefeld einen Wert einzugeben. Beim Klicken auf einen ebenfalls definierten Button wird die Funktion `BreiteFestlegen()` aufgerufen, die die Fensterbreite auf den eingegebenen Wert setzt. Beachten Sie beim Neufestsetzen der Fensterbreite die Bildschirmgröße. Diese können Sie mit dem `Screen`-Objekt ermitteln. Beachten Sie auch, dass der Anzeigebereich eines Fensters um einiges kleiner sein muss als das Fenster selbst, da das Fenster in der Regel noch Elemente wie Titelleiste, Menüleiste usw. enthält. Die absolute Breite eines Fensters können Sie mit `outerWidth` auslesen bzw. setzen.

**Beachten Sie:** Unter Opera 5.12 können Sie die Eigenschaft `innerWidth` nur auslesen, jedoch nicht setzen. Im MS Internet Explorer können Sie mit `document.body.offsetWidth` die innere Fensterbreite annähernd bestimmen. Zu diesem Zeitpunkt muss jedoch das `Body`-Element bereits vorhanden sein, d.h. es ist sinnvoll, die Ermittlung dieser Eigenschaft vom Event-Handler `onLoad` abhängig zu machen.

#### Lesen *locationbar*

Speichert die Information, ob ein Fenster eine eigene Adresszeile hat. Stellt selbst ein Objekt dar, das eine Eigenschaft hat, nämlich die Eigenschaft `visible` (= *sichtbar*). Enthält für diese Eigenschaft den Wert `true`, wenn das Fenster eine Adresszeile hat, und den Wert `false`, wenn es keine hat. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Fensterchen = window.open("datei.htm", "Zweitfenster", "width=300,height=200");
function Adress_Zeile() {
  if(Fensterchen.locationbar && Fensterchen.locationbar.visible == false) {
    Fensterchen.close();
    Neufenster = window.open("datei.htm", "Zweitfenster",
"width=300,height=200,location");
    Neufenster.focus();
  }
}
//-->
</script>
</head><body>
<a href="javascript:Adress_Zeile()">Adresszeile?</a>
</body></html>
```

Das Beispiel öffnet zunächst ein Zweitfenster ohne Adresszeile. Beim Klicken auf den Verweis wird die Funktion `Adress_Zeile()` aufgerufen. Diese Funktion prüft, ob der Browser das Objekt `locationbar` kennt und ob das Fenster keine Adresszeile hat. Wenn es keine hat, wird es geschlossen. Stattdessen wird ein neues Fenster geöffnet, wieder mit den gleichen Werten wie das alte, jedoch zusätzlich mit der Angabe `location`. Näheres dazu siehe `open()`.

**JS** **N** **Lesen** **menubar**  
**1.2** **4.0**

Speichert die Information, ob ein Fenster eine eigene Menüleiste mit den Browser-Menübefehlen hat. Stellt selbst ein Objekt dar, das eine Eigenschaft hat, nämlich die Eigenschaft `visible` (= *sichtbar*). Enthält für diese Eigenschaft den Wert `true`, wenn das Fenster eine Adresszeile hat, und den Wert `false`, wenn es keine hat. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Hinweis() {
  if(window.menubar && window.menubar.visible == true)
    alert("Um die Seite zu editieren, aus dem Menue File den Befehl Edit Page
    waehlen");
}
//-->
</script>
</head><body>
<a href="javascript:Hinweis()">Diese Seite editieren</a>
</body></html>
```

Das Beispiel enthält einen Verweis, das dem Anwender anbietet, die angezeigte Seite zu editieren. Wenn er den Verweis anklickt, wird eine Funktion `Hinweis()` aufgerufen. Diese prüft, ob der Browser das Objekt `menubar` kennt und ob für das aktuelle Fenster eine Menüleiste verfügbar ist. Wenn ja, wird mit `alert()` ein Hinweis ausgegeben, was der Anwender tun muss, um die Seite zu editieren.

**JS** **N** **Lesen** **name**  
**1.0** **2.0** **3.0** **Ändern**

Speichert den Namen eines Fensters. Bei Netscape 2.0 nur auslesbar, ab späteren Versionen auch änderbar. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Fenster = window.open("datei.htm","KleinesFenster","width=200,height=200");
function Fenstername() {
  alert("Name des kleinen Fensters: " + Fenster.name);
  neu = prompt("Vergeben Sie einen neuen Fenstername", "Name");
  Fenster.name = neu;
  alert("Der Name des Fensters lautet jetzt: " + Fenster.name);
}
//-->
</script>
</head><body>
<a href="javascript:Fenstername()">Name des Fensters</a>
</body></html>
```

Das Beispiel lädt beim Einlesen der Datei ein zweites kleines Fenster. Der Name dieses Fensters lautet `KleinesFenster`. Beachten Sie, dass die Eigenschaft `name` sich auf diesen Namen bezieht, nicht etwa auf den Variablennamen, in dem die Fensterinstanz gespeichert wird (im Beispiel die

Variable Fenster). Beim Klicken auf einen Verweis wird die Funktion `Fenstername()` aufgerufen. Darin wird dem Anwender zunächst mit `alert()` der aktuelle Fenstername des kleinen Fensters ausgegeben. Dann kann er einen neuen Namen eingeben. Abschließend wird der neue Name angezeigt.

**Beachten Sie:** Ein mit JavaScript gesetzter Fenstername bleibt so lange erhalten, wie das Fenster geöffnet ist bzw. bis er erneut geändert wird. Sie können aus diesem Grund den Fenstername zum Speichern von Variableninhalten verwenden, z.B., um Variablen von einer Datei an eine später aufgerufene Datei zu übergeben. Berücksichtigen Sie dabei jedoch, dass der Fenstername in vielen Browsern lediglich Buchstaben, Zahlen und den Unterstrich enthalten darf. Sonderzeichen und Umlaute dagegen sind nicht erlaubt. Eine Ausnahme bildet Opera 5. Hier bleibt ein mit JavaScript gesetzter Fenstername nur so lange gültig, wie das ändernde Dokument in diesem Fenster geladen ist. Lesen Sie zu dem Thema auch im Online-Angebot von SELFHTML ([selfhtml.teamone.de](http://selfhtml.teamone.de)) aktuell den Feature-Artikel *Javascript-Wertübergabe zwischen verschiedenen HTML-Dokumenten* von Hatto von Hatzfeld.



Speichert die Gesamthöhe eines Fensters, inklusive Titelleiste, Menüleiste, Statuszeile usw. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert("Gesamthoehe dieses Fensters: " + window.outerHeight + "  Pixel");
  window.outerHeight = 300;
  alert("Gesamthoehe jetzt: " + window.outerHeight + "  Pixel");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel gibt beim Einlesen zunächst die Höhe des aktuellen Fensters aus. Dann setzt es die Höhe neu fest, und zwar auf 300 Pixel. Anschließend wird die neue Fensterhöhe zur Kontrolle mit `alert()` nochmals ausgegeben.

**Beachten Sie:** Unter Opera 5.12 können Sie die Eigenschaft `outerHeight` nur auslesen, jedoch nicht setzen.



Speichert die Gesamtbreite eines Fensters, inklusive Fensterränder. Das Beispiel fragt beim Einlesen der Datei ab, ob die Breite des aktuellen Fensters mehr als 640 Pixel beträgt. Wenn ja, wird die Breite auf 640 Pixel festgesetzt.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(window.outerWidth > 640)
  window.outerHeight = 640;
//-->
```



```

</script>
</head><body>
</body></html>

```

**Beachten Sie:** Unter Opera 5.12 können Sie die Eigenschaft `outerWidth` nur auslesen, jedoch nicht setzen.

### JS 1.1 N 3.0 Lesen *pageXOffset*

Speichert die aktuelle horizontale Position innerhalb der Seite, gemessen am linken Fensterrand. So lässt sich ermitteln, wie weit der Anwender bereits nach rechts gescrollt hat. Sinnvoll in Verbindung der Verwendung mit `scrollBy()` oder `scrollTo()`. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Set0() {
  if(parent.frames[1].pageXOffset > 0)
    parent.frames[1].scrollTo(0,parent.frames[1].pageYOffset);
}
//-->
</script>
</head><body>
<a href="javascript:Set0()">nach links</a>
</body></html>

```

Im Beispiel wird angenommen, es gibt ein anderes Frame-Fenster, das einen überbreiten Inhalt hat, sodass der Anwender nach rechts scrollen muss, um alles zu sehen. Mit dem Verweis, der in der aktuellen Datei notiert ist, lässt sich das andere Frame-Fenster wieder ganz nach links scrollen. Dazu wird die aktuelle horizontale Scroll-Position der Eigenschaft `pageXOffset` abgefragt. Wenn sie größer 0 ist, hat der Anwender nach rechts gescrollt. In diesem Fall wird die Methode `scrollTo()` aufgerufen, um wieder ganz nach links zu scrollen. Die vertikale Position bleibt dabei erhalten, da als Parameter die Eigenschaft `pageYOffset` übergeben wird. Zur Adressierung von Frame-Fenstern in JavaScript lesen Sie den Abschnitt 20.3 über das Objekt `frames`.

**Beachten Sie:** Im MS Internet Explorer ab Version 4 ist die horizontale Scroll-Position in der Eigenschaft `document.body.scrollLeft` gespeichert.

### JS 1.2 N 4.0 Lesen *pageYOffset*

Speichert die aktuelle vertikale Position innerhalb der Seite, gemessen am Dokumentanfang. So lässt sich ermitteln, wie weit der Anwender bereits nach unten gescrollt hat. Sinnvoll ist die Verwendung in Verbindung mit `scrollBy()` oder `scrollTo()`. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
for(var i=0;i<100;i++)
  document.write("Viel Text<br>");
function Position() {

```

```

    alert("pageYOffset: " + window.pageYOffset + " Pixel");
}
//-->
</script>
<a href="javascript:Position()">Y-Position</a>
</body></html>

```

Im Beispiel schreibt ein JavaScript zu Testzwecken 100-mal eine HTML-Zeile mit dem Text *Viel Text* in das Dokumentfenster. Das veranlasst den Browser, entsprechend weit nach unten mit zu scrollen. Am Ende wird ein Verweis notiert. Bei dessen Anklicken wird die Funktion `Position()` aufgerufen. Sie gibt mit `alert()` aus, wie weit die aktuelle Scroll-Position vom Dokumentanfang entfernt ist.

**Beachten Sie:** Im MS Internet Explorer ab Version 4 ist die vertikale Scroll-Position in der Eigenschaft `document.body.scrollTop` gespeichert.

### JS 1.2 N 4.0 Lesen *personalbar*

Speichert die Information, ob ein Fenster eine eigene »Personal«-Leiste hat. Das ist bei Netscape 4.x die Leiste unterhalb der Adresszeile, die den Direktzugriff auf persönliche Lieblingsadressen ermöglicht. Stellt selbst ein Objekt dar, das eine Eigenschaft hat, nämlich die Eigenschaft `visible` (= *sichtbar*). Enthält für diese Eigenschaft den Wert `true`, wenn das Fenster eine Personal-Leiste hat, und den Wert `false`, wenn es keine hat. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function PersonalCheck() {
    if (window.personalbar)
        if(window.personalbar.visible == false)
            alert("Haben Sie denn keine Lieblings-Adressen?");
        else
            alert("Aha, Sie nutzen also die Personal Bar!");
    }
//-->
</script>
</head><body>
<a href="javascript:PersonalCheck()">Persoenlichkeitstest</a>
</body></html>

```

Das Beispiel enthält einen Verweis. Wenn der Anwender den Verweis anklickt, wird eine Funktion `PersonalCheck()` aufgerufen. Diese prüft, ob der Browser das Objekt `personalbar` kennt und ob für das aktuelle Fenster eine Personal-Leiste verfügbar ist. Je nach Ergebnis wird dem Anwender mit `alert()` eine »persönliche« Meldung ausgegeben.

### JS 1.2 N 4.0 Lesen *scrollbars*

Speichert die Information, ob ein Fenster Scroll-Leisten hat. Stellt selbst ein Objekt dar, das eine Eigenschaft hat, nämlich die Eigenschaft `visible` (= *sichtbar*). Enthält für diese Eigenschaft den Wert `true`, wenn das Fenster Scroll-Leisten hat, und den Wert `false`, wenn es keine hat. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Neufenster = window.open("datei.htm", "AnderesFenster","width=400,height=250");
function Test() {
    if(Neufenster.scrollbars && Neufenster.scrollbars.visible == false)
        Neufenster.close();
}
//-->
</script>
</head><body>
<a href="javascript:Test()">Kleiner Scroll-Test</a>
</body></html>

```

Das Beispiel lädt beim Einlesen der Datei ein zweites, kleineres Fenster, das anschließend unter `Neufenster` ansprechbar ist. Die Datei selbst enthält einen Verweis. Wenn der Anwender den Verweis anklickt, wird eine Funktion `Test()` aufgerufen. Diese prüft, ob der Browser das Objekt `scrollbars` kennt und ob für das zweite Fenster Scroll-Leisten verfügbar sind. Wenn nicht, wird das Fenster mit `close()` geschlossen.

#### JS 1.2 N 4.0 Lesen **statusbar**

Speichert die Information, ob ein Fenster eine eigene Statuszeile hat. Stellt selbst ein Objekt dar, das eine Eigenschaft hat, nämlich die Eigenschaft `visible` (= *sichtbar*). Enthält für diese Eigenschaft den Wert `true`, wenn das Fenster eine Statuszeile hat, und den Wert `false`, wenn es keine hat. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Statustext(Text)
{
    if(window.statusbar && window.statusbar.visible == true) window.defaultStatus =
Text;
    else alert(Text);
}
//-->
</script>
</head><body>
<form name="Eingabe" action="">
<input type="text" name="Feld">
<input type="button" value="Start"
onClick="Statustext(document.Eingabe.Feld.value)">
</form>
</body></html>

```

Im Beispiel wird ein Formular definiert. Der Anwender kann einen Text eingeben. Beim Klick auf den Button wird die Funktion `Statustext()` aufgerufen. Die Funktion ermittelt, ob der Browser das Objekt `scrollbars` kennt und ob das aktuelle Fenster eine Statuszeile hat. Wenn ja,

wird der im Eingabefeld eingegebene Text als Standardtext der Statuszeile gesetzt. Wenn nein, wird der eingegebene Text nur mit `alert()` angezeigt.



**status**

Speichert oder setzt den aktuellen Inhalt der Statuszeile. Das Beispiel zeigt einen Verweis. Beim Überfahren des Verweistextes mit der Maus erscheint in der Statuszeile der Text: das ist ein Verweis. Wichtig ist bei solchen Konstrukten die Anweisung `return true` am Ende.

```
<html><head><title>Test</title>
</head><body>
<a href="datei.htm" onmouseover="status='Das ist ein Verweis';return
true;">Verweis</a>
</body></html>
```



**toolbar**

Speichert die Information, ob ein Fenster eine eigene Button-Leiste (mit Buttons für »Vor«, »Zurück« usw.) hat. Stellt selbst ein Objekt dar, das eine Eigenschaft hat, nämlich die Eigenschaft `visible` (= *sichtbar*). Enthält für diese Eigenschaft den Wert `true`, wenn das Fenster eine Button-Leiste hat, und den Wert `false`, wenn es keine hat. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
if(self.toolbar && self.toolbar.visible == false)
  document.write("<a href='javascript:history.back()'>Back</a>");
//-->
</script>
</body></html>
```

Das Beispiel enthält innerhalb des Dateikörpers einen Script-Bereich. Darin wird überprüft, ob das Fenster eine eigene Button-Leiste hat. Wenn nicht, wird mit `document.write()` ein HTML-Konstrukt an die Stelle im Dokument geschrieben, die den Button »Zurück« als HTML-Verweis schreibt (siehe auch `history.back()`).



**alert()**

Gibt Text in einem Dialogfenster aus. Erwartet eine Zeichenkette, eine Zahl oder ein Objekt als Parameter. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:alert('Verweisziel noch nicht erreichbar')">Unsere Erfolge</a>
</body></html>
```

Das Beispiel führt beim Anklicken eines Verweises keinen Sprung zu einer anderen Seite aus, sondern gibt stattdessen eine Dialogbox mit dem Hinweis "Verweisziel noch nicht erreichbar" aus. Sie können der `alert()`-Methode feste Zeichenketten übergeben wie im Beispiel, aber

auch Variablen. Dann wird der Inhalt der Variablen ausgegeben. Auch zusammengesetzte Ketten dürfen Sie übergeben, etwa: `alert("Ihr Browser" + navigator.userAgent)`

**Beachten Sie:** Intern wandelt die `alert()`-Methode den übergebenen Parameter in eine Zeichenkette um. So kann es vorkommen, dass sich ein Wert mit `alert()` ausgeben lässt, jedoch keine Zeichenkettenoperationen mit dem ausgegebenen Wert ausführbar sind, da es sich außerhalb der Ausgabe nicht um eine Zeichenkette handelt. Der Rückgabewert der Methode `alert()` ist stets `undefined`.

Das Aussehen des Mitteilungsfenster ist abhängig vom Betriebssystem und dem verwendeten Browser. Das bedeutet, Sie können die Mitteilungsbbox nur mittels Steuerzeichen formatieren (Kapitel 21.5.1). Weitere Formatierungsmöglichkeiten stehen Ihnen nicht zur Verfügung.

### **back()**

Entspricht bei normalen WWW-Seiten einem Klick auf den Zurück-Button im Browser. Bei Frames wird jedoch die letzte besuchte Seite aufgerufen, die nicht zum Frameset gehörte. Hier liegt der entscheidende Unterschied zur Methode `history.back`! Diese Methode erwartet keine Parameter. Das Beispiel realisiert einen HTML-Verweis, bei dessen Anklicken die Seite aufgerufen wird, die zuletzt das gesamte Anzeigefenster ausfüllte:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:window.back()">Zurück</a>
</body></html>
```

### **blur()**

Macht ein Fenster inaktiv. Das Gegenteil von `focus()`. Erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Neufenster = window.open("datei.htm", "AnderesFenster", "width=400,height=250");
//-->
</script>
</head><body>
<a href="javascript:Neufenster.focus()">Fenster nach vorne</a><br>
<a href="javascript:Neufenster.blur()">Fenster nach hinten</a>
</body></html>
```

Im Beispiel wird beim Einlesen der HTML-Datei ein zweites, kleines Fenster geöffnet. Die HTML-Datei selbst enthält zwei Verweise. Beim Anklicken des ersten Verweises wird das zweite Fenster aktiviert – `focus()`. Beim Anklicken des zweiten Verweises wird es deaktiviert und je nach Fensterkonstellation von dem Hauptfenster überdeckt – `blur()`.

### **captureEvents()**

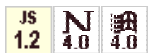
Überwacht Anwenderereignisse im angegebenen Fenster. Erwartet als Parameter eine Folge von Ereignissen, die überwacht werden sollen. Folgende Ereignisse lassen sich überwachen: `Event.ABORT` `Event.BLUR` `Event.CHANGE` `Event.CLICK` `Event.DBLCLICK` `Event.DRAGDROP` `Event.`

ERROR Event.FOCUS Event.KEYDOWN Event.KEYPRESS Event.KEYUP Event.LOAD Event.MOUSEDOWN Event.MOUSEMOVE Event.MOUSEOUT Event.MOUSEOVER Event.MOUSEUP Event.MOVE Event.RESET Event.RESIZE Event.SELECT Event.SUBMIT Event.UNLOAD. Es handelt sich dabei um spezielle Notationsvarianten entsprechender Event-Handler ohne das »on« davor, dafür mit dem Ansprechen des Event-Objekts davor. So entspricht etwa Event.MOUSEOVER dem Event-Handler onmouseover. Sie können mehrere Ereignisse überwachen. Trennen Sie die Event-Namen dabei durch | (Querstriche). Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
window.captureEvents(Event.KEYPRESS);
window.onkeypress = Ausgabe;
function Ausgabe(Ereignis) {
    alert("Sie haben die Taste mit dem Wert " + Ereignis.which + " gedrueckt");
}
//-->
</script>
</head><body>
Druecken Sie irgendwelche Tasten!
</body></html>
```

Im Beispiel wird der Event KEYPRESS (*Taste gedrückt*) überwacht. Wenn der Anwender eine Taste drückt, wird die Funktion Ausgabe() aufgerufen, die mit alert() ausgibt, welche Taste gedrückt wurde. Die Syntax beim Funktionsaufruf ist dabei abweichend von der üblichen. Ein zusammenhängendes Beispiel mit Erläuterungen hierzu finden Sie im Kapitel 21 *Dynamisches HTML* im Abschnitt 21.4.7 über das Event-Objekt bei Netscape.

Auch Netscape 6 interpretiert das vorliegende Beispiel. Für diesen Browser ist es jedoch notwendig, nach der Bestätigung des Meldefensters dem Fenster mittels Klick erneut den Fokus zu geben.



### clearInterval()

Bricht einen Endlosvorgang ab, der mit setInterval() begonnen wurde. Erwartet als Parameter die Variable, in der der Aufruf von setInterval() gespeichert wurde. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var aktiv = window.setInterval("Farbe()",1000);
var i = 0, farbe = 1;
function Farbe() {
    if(farbe==1)
    { document.bgColor="yellow"; farbe=2; }
    else
    { document.bgColor="aqua"; farbe=1; }
    i = i + 1;
    if(i >= 10)
        window.clearInterval(aktiv);
}
```

```

}
//-->
</script>
</body></html>

```

Das Beispiel definiert mit `setInterval()`, dass die Funktion `Farbe()` alle 1000 Millisekunden, also jede Sekunde einmal aufgerufen wird. Der Aufruf wird in der Variablen `aktiv` gespeichert. Im Beispiel wechselt die Funktion `Farbe()` bei jedem Aufruf die Hintergrundfarbe der Datei (`document.bgColor`). Gleichzeitig wird ein Zähler hochgezählt. Wenn er größer oder gleich 10 ist, wird die Methode `clearInterval()` aufgerufen, die den Endlosvorgang stoppt. Dabei wird die zuvor definierte Variable `aktiv` als Parameter übergeben.

### `clearTimeout()`

Bricht einen Timeout ab, der mit der Methode `setTimeout()` gestartet wurde. Erwartet als Parameter die Variable, in der der Aufruf von `setTimeout()` gespeichert wurde. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var i = 0;
function Zaehlen() {
    i = i + 1;
    aktiv = window.setTimeout("Zaehlen()",1000);
}
function Aufhoeren() {
    window.clearTimeout(aktiv);
    alert(i + " Sekunden");
}
aktiv = window.setTimeout("Zaehlen()",1000);
//-->
</script>
</head><body>
<form action="">
<input type="button" value="Klick" onClick="Aufhoeren()">
</form>
</body></html>

```

Das Beispiel startet beim Einlesen der Datei mit `setTimeout()` einen Zählvorgang. Dabei wird alle 1000 Millisekunden, also 1 Mal pro Sekunde, die Funktion `Zaehlen()` aufgerufen. Diese zählt die Variable `i` hoch. In der Datei wird ein Button definiert. Wenn der Anwender auf den Button klickt, wird die Funktion `Aufhoeren()` aufgerufen. Diese Funktion stoppt mit `clearTimeout()` den Timeout und gibt mit `alert()` die Anzahl der gezählten Sekunden aus. Beim Aufruf von `clearTimeout()` wird die zuvor bei `setTimeout()` definierte globale Variable `aktiv` als Parameter übergeben.




**close()**

Schließt ein Fenster. Erwartet keine Parameter. Das Beispiel öffnet beim Einlesen der Datei ein zweites Fenster namens *Info*. In der Beispieldatei wird ein Button definiert. Wenn der Anwender auf den Button klickt, wird das zweite Fenster geschlossen.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Info = window.open("datei.htm", "Zweitfenster");
//-->
</script>
</head><body>
<form action="">
<input type="button" value="Fenster zu" onClick="Info.close()">
</form>
</body></html>
```

**Beachten Sie:** Die Methode `close()` unterliegt der Sicherheitskonzeption von JavaScript. Das bedeutet, dass sich ein Fenster, sobald es eine History besitzt (weil der Anwender darin mehrere Seiten aufgerufen hat), nicht mehr ohne eine Browser-Nachfrage schließen lässt. Ein Unterdrücken dieser Abfrage ist nicht möglich.




**confirm()**

Blendet ein Dialogfenster mit zwei Buttons für  und  ein. Sinnvoll, um vom Anwender eine Entscheidung zu erzwingen, die im weiteren Programmablauf verarbeitet wird. Erwartet als Parameter einen Aufforderungstext für die Ja/Nein-Entscheidung. Liefert als Ergebnis zurück, wie sich der Anwender entschieden hat. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Check = confirm("Wollen Sie diese Seite wirklich sehen?");
if(Check == false) history.back();
//-->
</script>
</head><body>
</body></html>
```

Im Beispiel wird der Anwender beim Einlesen der Datei gefragt, ob er die Seite wirklich sehen will. Der Rückgabewert von `confirm()` wird in der Variablen `Check` gespeichert. Wenn der Anwender auf  klickt, enthält `Check` den Wert `true`. Es passiert nichts weiter und die Seite wird geladen. Wenn er »Abbrechen« wählt, erhält `Check` den Wert `false`. Für diesen Fall ist die Anweisung `history.back()` notiert, die den Anwender auf die Seite zurückbefördert, von der er kam.

Das Aussehen des Auswahlfensters ist abhängig vom Betriebssystem und dem verwendeten Browser. Das bedeutet, Sie können die Auswahl nur mittels Steuerzeichen formatieren (Kapitel 21.5.1). Weitere Formatierungsmöglichkeiten stehen Ihnen nicht zur Verfügung.



***disableExternalCapture()***

Verbietet anderen Fenstern, Ereignisse im aktuellen Fenster zu überwachen. Vor allem als Schutz gedacht, falls fremde Seiten die eigene Seite in ein Frameset laden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
disableExternalCapture();
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel startet beim Einlesen den Befehl `disableExternalCapture()`. Dadurch können fremde Seiten keine Anwenderereignisse in dieser Datei überwachen, falls die Datei in ein fremdes Frameset geladen wird.

***enableExternalCapture()***

Erlaubt anderen Fenstern, Ereignisse im aktuellen Fenster zu überwachen. So lassen sich von anderen Fenstern eines Framesets aus Anwenderereignisse im aktuellen Fenster überwachen. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
enableExternalCapture();
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel startet beim Einlesen den Befehl `enableExternalCapture()`. Dadurch können fremde Seiten Anwenderereignisse in dieser Datei überwachen, falls die Datei Teil eines Framesets ist.

***find()***

Durchsucht den Inhalt eines Fensters nach einem Text. Entspricht in der Funktionalität dem Menübefehl »Suchen«. Die Suche bezieht sich nur auf die aktuelle Seite.

Erwartet folgende Parameter:

1. *Text* = Text, nach dem gesucht werden soll.
2. *Groß-/Kleinschreibung beachten* = `true` für ja oder `false` für nein übergeben.
3. *Suchrichtung rückwärts* = `true` für ja oder `false` für nein übergeben.

Die beiden letzten Parameter sind optional. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form action="">
<input type="text" name="SuchNach">
```

```



```

Im Beispiel steht am Anfang der Datei ein Formular mit einem Eingabefeld zur Eingabe eines Suchbegriffs. Das Formular enthält außerdem einen Button. Bei dessen Anklicken wird der weitere Inhalt der Datei nach dem Begriff durchsucht, den der Anwender in dem Eingabefeld eingegeben hat.

**Beachten Sie:** Netscape 6.1 kennt diese Methode nicht (mehr).



**focus()**

Macht ein Fenster zum aktiven Fenster. Sinnvoll bei Verwendung von Zweitfenstern, aber auch bei Frame-Fenstern. Erwartet keine Parameter. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function FrameVerweis(Ziel)
{
    parent.frames[1].location.href = Ziel;
    parent.frames[1].focus();
}
//-->
</script>
</head><body>
<a href="javascript:FrameVerweis('../location.htm')">Neue Seite</a>
</body></html>

```

Das Beispiel nimmt an, dass die aktuelle Datei Teil eines Framesets ist. Die Datei enthält einen Verweis, dessen Ziel in einem anderen Frame-Fenster angezeigt werden soll. Beim Anklicken wird aber nicht nur das Verweisziel in das andere Frame-Fenster geladen, sondern dieses wird zugleich das aktive Fenster. So kann der Anwender beispielsweise gleich mit den Pfeiltasten darin scrollen. Dazu wird beim Anklicken des Verweises eine Funktion `FrameVerweis()` aufgerufen. Diese Funktion lädt im ersten Befehl das als Parameter übergebene Verweisziel in den anderen Frame (`location.href`), und im zweiten Befehl setzt sie mit `focus()` das Ziel-Frame-Fenster aktiv. Zur Adressierung von Frame-Fenstern in JavaScript lesen Sie den Abschnitt 20.3 über das Objekt `frames`.



**forward()**

Entspricht bei normalen WWW-Seiten einem Klick auf den Vorwärts-Button im Browser. Bei Frames wird jedoch die nächste besuchte Seite aufgerufen, die nicht zum Frameset gehört. Bei Frames also unterschiedlich zu `history.forward()`! Erwartet keine Parameter. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<a href="javascript:window.forward()">Vor</a>
</body></html>

```

Das Beispiel realisiert einen HTML-Verweis, bei dessen Anklicken die Seite aufgerufen wird, die beim Vorwärtsgen in der History als nächste das gesamte Anzeigefenster ausfüllt.

### **handleEvent()**

Übergibt ein Ereignis an ein Element, das in der Lage ist, auf das Ereignis zu reagieren. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function clickHandler(Ereignis)
{ window.document.links[0].handleEvent(Ereignis); }
window.captureEvents(Event.CLICK);
window.onClick = clickHandler;
//-->
</script>
</head><body>
<a href="datei.htm" onClick="location.href='datei.htm'">Verweis</a>
</body></html>
```

Das Beispiel enthält einen Verweis, bei dessen Anklicken – ganz normal durch die href-Angabe – die Datei *datei.htm* aufgerufen wird. Im Beispiel ist es jedoch so, dass *datei.htm* sofort aufgerufen wird, sobald der Anwender irgendwo im Browser-Fenster klickt. Dazu wird beim Einlesen der Datei mit der `captureEvents()`-Methode das Mausklick-Ereignis überwacht. Wenn das Ereignis eintritt, wird die Funktion `ClickHandler()` aufgerufen. Diese übergibt mit `handleEvent()` das Ereignis an den ersten im Dokument definierten Verweis weiter (`document.links[0]`). In dem entsprechenden Verweis ist dazu ein Event-Handler `onClick=` notiert. Dieser ist in der Lage, das übergebene Ereignis zu verarbeiten. Als Befehl wird mit `location.href` die gleiche Datei aufgerufen wie beim Anklicken des Verweises selbst.

**Beachten Sie:** Netscape 6.1 kennt diese Methode nicht (mehr).

### **home()**

Entspricht einem Klick auf den -Button im Browser. Das Beispiel definiert einen Verweis, bei dessen Anklicken die Seite aufgerufen wird, die der Anwender als Start-Homepage eingestellt hat.

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:home()">Nach hause</a>
</body></html>
```

### **moveBy()**

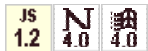
Verschiebt ein Fenster um so viele Pixel wie angegeben. Erwartet folgende Parameter:

1. *x-Wert* = um wie viele Pixel das Fenster nach links/rechts verschoben werden soll. Negative Werte verschieben nach links, positive nach rechts.
2. *y-Wert* = um wie viele Pixel das Fenster nach oben/unten verschoben werden soll. Negative Werte verschieben nach oben, positive nach unten.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Fenster = window.open("datei.htm","Zweitfenster","width=200,height=200");
function Bewege() {
  Fenster.moveBy(200,100);
  Fenster.focus();
}
//-->
</script>
</head><body>
<a href="javascript:Bewege()">Fenster bewegen</a>
</body></html>
```

Das Beispiel lädt beim Einlesen der Datei ein kleines Zweitfenster mit einer anderen Datei. Das Hauptfenster enthält einen Verweis. Beim Anklicken des Verweises wird die Funktion `Bewege()` aufgerufen, die das kleine Fenster um 200 Pixel nach rechts und um 100 Pixel nach unten verschiebt.



***moveTo()***

Verschiebt ein Fenster auf die anzugebende Position. Erwartet folgende Parameter:

1. *x-Wert* = Links-Wert für die neue obere linke Ecke des Fensters.
2. *y-Wert* = Oben-Wert für die neue obere linke Ecke des Fensters.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Fenster = window.open("datei.htm","Zweitfenster","width=200,height=200");
function Bewege() {
  Fenster.moveTo(screen.width-200,screen.height-200);
  Fenster.focus();
}
//-->
</script>
</head><body>
<a href="javascript:Bewege()">Fenster bewegen</a>
</body></html>
```

Das Beispiel lädt beim Einlesen der Datei ein kleines Zweitfenster mit einer anderen Datei. Das Zweitfenster hat eine Breite und eine Höhe von 200 Pixeln. Das Hauptfenster enthält einen Verweis. Beim Anklicken des Verweises wird die Funktion `Bewege()` aufgerufen, die das kleine Fenster in die rechte untere Ecke des Bildschirms verschiebt. Die Pixelangaben für die neue linke obere Ecke des Fensters werden dabei mit Hilfe des `screen`-Objekts gewonnen, das mit `screen.width` die Bildschirmbreite und mit `screen.height` die Bildschirmhöhe liefert.



Öffnet ein neues Fenster. Erwartet mindestens zwei, optional auch drei Parameter:

1. **URI** = Zieladresse einer Datei, die in das neue Fenster geladen werden soll. Wenn sich die Datei im gleichen Verzeichnis befindet, genügt der Dateiname. Ansonsten relative Pfadnamen oder absolute http-Adressen angeben. Bei Netscape darf dies auch eine leere Zeichenkette sein (öffnet ein leeres Fenster), was beim MS Internet Explorer 4 und Opera 5.12 allerdings zu einer Fehlermeldung führt. Anstelle eines URI können Sie auch mit `about:blank` eine leere Datei in das Fenster laden.
2. **Fenstername** = Ein Name, der aus Buchstaben, Ziffern und Unterstrich bestehen darf. Unter diesem Namen können beispielsweise Verweise ihre Ziele mit `<a href="ziel.htm" target="NameDesFensters">` in das erzeugte Fenster laden. Gültige Fenstername sind auch `_blank`, `_parent`, `_self` und `_top`. Opera 5.12 öffnet jedoch in jedem Fall die Datei im gleichen Fenster. Netscape 6.x interpretiert die Angabe `_parent` nicht.
3. (optional) **Angaben zum Aussehen des Fensters** = Eine Zeichenkette, in der Sie die Größe und die Eigenschaften des Fensters festlegen können. Mehrere Angaben sind durch Kommata zu trennen. Folgende Angaben sind möglich:

| Eigenschaft               | Wertzuweisung | Status                                                                                                                                                                                                                                                      | Bedeutung                                                                                                                                                                                                                                                                      |
|---------------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dependent=</code>   | yes/no        |                                                                                         | Wenn ja (yes), wird das Fenster geschlossen, wenn sein Elternfenster geschlossen wird. Wenn nein (no = Voreinstellung), bleibt das Fenster erhalten, wenn sein Elternfenster geschlossen wird.                                                                                 |
| <code>height=</code>      | [Pixel]       |    | Höhe des neuen Fensters in Pixeln, z.B. <code>height=200</code> .                                                                                                                                                                                                              |
| <code>hotkeys=</code>     | yes/no        |                                                                                       | Wenn nein (no), werden Tastaturbefehle zum Steuern des Browsers in dem Fenster deaktiviert. Wenn ja (yes = Voreinstellung), bleiben Tastaturbefehle des Browsers in dem Fenster gültig.                                                                                        |
| <code>innerHeight=</code> | [Pixel]       |                                                                                       | Höhe des Anzeigebereichs des neuen Fensters in Pixeln, z.B. <code>innerHeight=200</code> .                                                                                                                                                                                     |
| <code>innerWidth=</code>  | [Pixel]       |                                                                                       | Breite des Anzeigebereichs des neuen Fensters in Pixeln, z.B. <code>innerWidth=200</code> .                                                                                                                                                                                    |
| <code>left=</code>        | [Pixel]       |    | Horizontalwert der linken oberen Ecke des neuen Fensters in Pixeln, z.B. <code>left=100</code> .                                                                                                                                                                               |
| <code>location=</code>    | yes/no        |    | Wenn ja (yes), erhält das Fenster eine eigene Adresszeile. Wenn nein (no), erhält es keine Adresszeile. Voreinstellung ist no, beim Internet Explorer jedoch nur, wenn die Optionenzeichenkette mindestens eine Option enthält. Netscape 6.1 interpretiert diese Angabe nicht. |

| Eigenschaft | Wertzuweisung | Status                                                                                                                                                                                                                                                      | Bedeutung                                                                                                                                                                                                                                           |
|-------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| menubar=    | yesno         |          | Wenn ja (yes), erhält das Fenster eine eigene Menüleiste mit Browser-Befehlen. Wenn nein (no), erhält es keine Menüleiste. Voreinstellung ist no, beim Internet Explorer jedoch nur, wenn die Optionenzeichenkette mindestens eine Option enthält.  |
| resizable=  | yesno         |          | Wenn ja (yes), kann der Anwender das Fenster in der Größe verändern. Wenn nein (no), kann er die Fenstergröße nicht ändern. Voreinstellung ist no, beim Internet Explorer jedoch nur, wenn die Optionenzeichenkette mindestens eine Option enthält. |
| screenX=    | [Pixel]       |                                                                                           | Horizontalwert der linken oberen Ecke des neuen Fensters in Pixeln, z.B. screenX=100.                                                                                                                                                               |
| screenY=    | [Pixel]       |                                                                                           | Vertikalwert der linken oberen Ecke des neuen Fensters in Pixeln, z.B. screenY=30.                                                                                                                                                                  |
| scrollbars= | yesno         |          | Wenn ja (yes), erhält das Fenster Scroll-Leisten. Wenn nein (no), kann der Anwender in dem Fenster nicht scrollen. Voreinstellung ist no, beim Internet Explorer jedoch nur, wenn die Optionenzeichenkette mindestens eine Option enthält.          |
| status=     | yesno         |    | Wenn ja (yes), erhält das Fenster eine eigene Statuszeile. Wenn nein (no), erhält es keine Statuszeile. Voreinstellung ist no, beim Internet Explorer jedoch nur, wenn die Optionenzeichenkette mindestens eine Option enthält.                     |
| toolbar=    | yesno         |    | Wenn ja (yes), erhält das Fenster eine eigene Werkzeugleiste. Wenn nein (no), erhält es keine Werkzeugleiste. Voreinstellung ist no, beim Internet Explorer jedoch nur, wenn die Optionenzeichenkette mindestens eine Option enthält.               |
| top=        | [Pixel]       |    | Vertikalwert der linken oberen Ecke des neuen Fensters in Pixeln, z.B. top=100.                                                                                                                                                                     |
| width=      | [Pixel]       |    | Breite des neuen Fensters in Pixeln, z.B. width=400.                                                                                                                                                                                                |

#### Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
F1 = window.open("datei.htm", "Fenster1", "width=310,height=400,left=0,top=0");
F2 = window.open("datei.htm", "Fenster2", "width=310,height=400,left=320,top=0");
self.focus();

```

```
self.close();
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel öffnet beim Einlesen der Datei zwei weitere Fenster so, dass sie nebeneinander angeordnet sind. Anschließend schließt sich das Hauptfenster selbst. Angenommen, in *datei.htm* (wird ins erste Fenster geladen) steht ein Verweis, dessen Ziel im zweiten Fenster angezeigt werden soll. Dazu können Sie notieren:

```
<a href="datei.htm" target="Fenster2">Verweistext</a>
```

Wichtig ist dabei die Angabe `target=`. Dort müssen Sie den Fensternamen angeben, den Sie bei der Definition des gewünschten Zielfensters vergeben haben – im Beispiel "Fenster2".

**Beachten Sie:** Die meisten Browser erlauben keine Fensterhöhe oder -breite, die kleiner als 100 px ist. Je nach Betriebssystem und Browser wird bei kleineren Fenstern eine Fehlermeldung (Zugriff verweigert) ausgegeben bzw. die jeweiligen Minimalwerte verwendet.

   **print()**

Druckt den Inhalt einer Seite aus. Entspricht dem Menübefehl zum Drucken im Browser. Erwartet keine Parameter. Das Beispiel definiert einen Verweis, mit dessen Hilfe der Anwender die Seite ausdrucken kann:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:window.print()">Diese Seite drucken</a>
</body></html>
```

**Beachten Sie:** Der MS Internet Explorer 5 unter Macintosh und Opera 5.12 interpretieren diese Methode nicht. Für den Internet Explorer 4.x gibt es einen Workaround, um das Ausdrucken mit JavaScript anzustoßen. Dazu muss ein ActiveX-Control eingebunden werden. Der JavaScript-Code dazu sieht folgendermaßen aus (Beispiel):

```
if(document.all && navigator.appVersion.substring(22,23)==4) {
  self.focus();
  var OLECMDID_PRINT = 6;
  var OLECMDEXECOPT_DONTPROMPTUSER = 2;
  var OLECMDEXECOPT_PROMPTUSER = 1;
  var WebBrowser = '<object id="WebBrowser1" width="0" height="0"
classid="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2"></object>';
  document.body.insertAdjacentHTML('beforeEnd',WebBrowser);
  WebBrowser1.ExecWB(OLECMDID_PRINT,OLECMDEXECOPT_DONTPROMPTUSER);
  WebBrowser1.outerHTML = '';
}
```

   **prompt()**

Zeigt ein Dialogfenster mit einem Eingabefeld, einem -Button und einem -Button an. Der Anwender kann in dem Eingabefeld etwas eingeben. Die Methode `prompt()` gibt

diesen eingegebenen Wert zurück. So lassen sich Anwendereingaben im Script weiterverarbeiten. Erwartet zwei Parameter:

1. *Aufforderungstext* = Text, der beschreibt, was der Anwender eingeben soll.
2. *Feldvorbelegung* = Text, mit dem das Eingabefeld vorbelegt wird. Wenn Sie ein leeres Eingabefeld wollen, übergeben Sie dafür eine leere Zeichenkette "".

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Check = prompt("Geben Sie Ihr Passwort fuer diese Seite ein","");
if(Check != "Rumpelstielz") history.back();
else alert("Sie haben Zutritt");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel bringt vor dem Anzeigen der HTML-Datei mit `prompt()` ein Dialogfenster auf den Bildschirm. Darin muss der Anwender ein Passwort eingeben. Die Eingabe wird in der Variablen `Check` gespeichert. Wenn der Wert dieser Variablen nach dem Beenden des Dialogfensters nicht gleich "Rumpelstielz" ist, wird der Anwender mit `history.back()` zu der Seite zurückgeschickt, von der er kam. Wenn das eingegebene Wort stimmt, erhält er mit `alert()` die Meldung, dass er Zutritt hat, und die Seite wird geladen.



### **releaseEvents()**

Beendet die Überwachung von Anwenderereignissen. Gegenstück zu `captureEvents()`. Erwartet als Parameter eine Folge von Ereignissen, die nicht mehr überwacht werden sollen. Gleiche Syntax wie bei `captureEvents()`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var i = 0;
window.captureEvents(Event.KEYPRESS);
window.onkeypress = Ausgabe;
function Ausgabe(Ereignis)
{
    alert("Sie haben die Taste mit dem Wert " + Ereignis.which + " gedrueckt");
    i = i + 1;
    if(i > 2)
        window.releaseEvents(Event.KEYPRESS);
}
//-->
</script>
</head><body>
Druecken Sie 3 mal irgendwelche Tasten!
</body></html>
```



Im Beispiel wird der Event `KEYPRESS` (*Taste gedrückt*) überwacht. Wenn der Anwender eine Taste drückt, wird die Funktion `Ausgabe()` aufgerufen, die mit `alert()` ausgibt, welche Taste gedrückt wurde. Die Funktion ruft sich bei Eintritt des Ereignisses selbst wieder auf, jedoch nur, solange der Zähler `i` kleiner als 3 ist. Danach wird das Überwachen der Tastaturereignisse mit der Methode `releaseEvents(Event.KEYPRESS)` beendet. Ein zusammenhängendes Beispiel mit Erläuterungen hierzu finden Sie im Kapitel 21 *Dynamisches HTML* im Abschnitt 21.4.7 über das Event-Objekt bei Netscape.

**Beachten Sie:** Netscape 6 interpretiert diese Methode nicht (mehr).



Verändert die Größe eines Fensters um so viele Pixel unten und rechts wie angegeben. Erwartet folgende Parameter:

1. *x-Wert* = um wie viele Pixel das Fenster rechts verkleinert/vergrößert werden soll. Negative Werte verkleinern, positive vergrößern.
2. *y-Wert* = um wie viele Pixel das Fenster unten verkleinert/vergrößert werden soll. Negative Werte verkleinern, positive vergrößern.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Kleiner() { window.resizeBy(-10,-10); }
//-->
</script>
</head><body>
<a href="javascript:Kleiner()">Mach kleiner</a>
</body></html>
```

Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird die Funktion `Kleiner()` aufgerufen. Diese Funktion verkleinert das dateieigene Fenster um 10 Pixel in der Breite und in der Höhe.



Verändert die Größe eines Fensters absolut. Erwartet folgende Parameter:

1. *x-Wert* = neue Breite des Fensters.
2. *y-Wert* = neue Höhe des Fensters.

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body onload="window.resizeTo(580,420)">
</body></html>
```

Das Beispiel setzt die Fenstergröße beim Einlesen der Datei auf eine Breite von 580 Pixeln und auf eine Höhe von 420 Pixeln.



**routeEvent()**

Übergibt ein Anwenderereignis an das oberste Objekt der Objekthierarchie, also an das `window`-Objekt. Vor dort aus wird das Ereignis immer weiter nach unten durchgereicht, bis sich ein Event-Handler findet, der das Ereignis verarbeitet. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Click1() {
  alert("Sie haben auf den Button geklickt");
}
function Click2(Ereignis)
{
  alert("Sie haben ins Fenster geklickt");
  window.routeEvent(Ereignis);
}
window.captureEvents(Event.CLICK);
window.onClick = Click2;
//-->
</script>
</head><body>
<form action="">
<input type="button" value="Button" onClick="Click1()">
</form>
</body></html>
```

Das Beispiel enthält einen Button, bei dessen Anklicken die Funktion `Click1()` aufgerufen wird. Diese Funktion gibt zur Kontrolle mit `alert()` aus, dass der Button angeklickt wurde. Im Beispiel ist es jedoch so, dass beim Einlesen der Datei mit der `captureEvents()`-Methode das Mausclick-Ereignis überwacht wird. Wenn das Ereignis eintritt, wird in jedem Fall die Funktion `Click2()` aufgerufen. Diese gibt zur Kontrolle aus, dass in das Fenster geklickt wurde. Anschließend übergibt sie mit `routeEvent()` das Ereignis an die Objekthierarchie. Wenn der Klick auf ein anderes Objekt erfolgt, das ebenfalls das Klick-Ereignis überwacht, wird die entsprechende Aktion ebenfalls ausgeführt. Im Beispiel hat dies zur Folge, dass der Anwender beim Klick auf den Button zwei Meldungen erhält: nämlich zuerst die, dass ins Fenster geklickt wurde, und danach die, dass auf den Button geklickt wurde. Ohne die `routeEvent()`-Methode würde er die zweite Meldung gar nicht erhalten, da das Klick-Ereignis vorher abgefangen worden wäre.

**Beachten Sie:** Netscape 6 interpretiert diese Methode nicht (mehr).




**scrollBy()**

Scrollt eine Seite automatisch um so viele Pixel nach oben oder unten oder links oder rechts wie angegeben. Erwartet folgende Parameter:

1. *x-Wert* = um wie viele Pixel nach rechts links/rechts gescrollt werden soll. Negative Werte scrollen nach links, positive nach rechts.
2. *y-Wert* = um wie viele Pixel nach oben/unten gescrollt werden soll. Negative Werte scrollen nach oben, positive nach unten.

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
for(var i=1; i<=100; i++)
  document.write("<br>Zeile " + i);
function NachOben() {
  var y=0;
  if (window.pageYOffset)
  else if (document.body && document.body.scrollTop) y=document.body.scrollTop;
  while(y > 0) {
    setTimeout("window.scrollTo(0,-10)",10);
    y = y - 10;
  }
}
//-->
</script>
<a href="javascript:NachOben()">nach oben scrollen</a>
</body></html>
```

Das Beispiel schreibt zu Testzwecken in einer Schleife mit `document.write()` 100 Zeilen in die Datei. Unterhalb davon wird ein HTML-Verweis notiert. Um zu diesem Verweis zu gelangen, muss der Anwender wegen der 100 voranstehenden Zeilen erst mal nach unten scrollen. Wenn er auf den Verweis klickt, wird automatisch langsam zum Seitenanfang zurückgescrollt. Dazu wird die Funktion `NachOben()` aufgerufen. Diese Funktion ermittelt mit `pageYOffset` bzw. im MS Internet Explorer mit `document.body.offsetTop` zunächst die vertikale Position innerhalb der Seite. Dann wird innerhalb einer `while`-Schleife so lange nach oben gescrollt, bis die Scrollposition bei 0 angelangt ist.

In dem Beispiel wird dazu nur der `y`-Wert jeweils um -10 Pixel verändert. So wird die Seite bei jedem Scroll-Befehl um 10 Pixel nach oben gescrollt. Der Verzögerungseffekt beim Scrollen kommt im Beispiel dadurch zustande, dass die Scroll-Befehle in die Methode `setTimeout()` eingebunden sind.



Scrollt zu einer bestimmten Position. Erwartet folgende Parameter:

1. *x-Wert* = Links-Wert für Position der linken oberen Ecke des Anzeigefensters.
2. *y-Wert* = Oben-Wert für Position der linken oberen Ecke des Anzeigefensters.

Ein Beispiel:

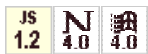
```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
function Pos100() { window.scrollTo(100,100) }
var Zeile = "<nobr>Zeile";
for(var i=1; i<=100; i++)
{
```

```

Zeile = Zeile + " Zeile";
document.write("<br>Zeile " + Zeile);
}
document.write("<a href='javascript:Pos100()''> Test</a></nobr>");
//-->
</script>
</body></html>

```

Das Beispiel schreibt zu Testzwecken in einer Schleife mit `document.write()` 100 Zeilen, wobei die Zeilen immer länger werden. Ganz am Ende wird ein Verweis notiert. Um dort hin zu gelangen, muss der Anwender nach unten und nach rechts scrollen. Wenn er auf den Verweis klickt, wird die Funktion `Pos100()` aufgerufen. Diese Funktion setzt die Scroll-Position neu, und zwar auf 100 Pixel von links und 100 Pixel von oben, bezogen auf den Dokumentanfang.



### *setInterval()*

Führt eine Anweisung oder den Aufruf einer Funktion in einem festgelegten Rhythmus immer wieder aus, solange, bis der Vorgang mit `clearInterval()` beendet wird. Erwartet zwei Parameter:

1. *Code* = Eine JavaScript-Anweisung, die wiederholt werden soll. Meistens ein Funktionsaufruf.
2. *Pause* = Wert in Millisekunden bis zum nächsten Ausführen.

Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var aktiv = window.setInterval("Farbe()",1000);
var i = 0, farbe = 1;
function Farbe() {
    if(farbe==1)
    { document.bgColor="yellow"; farbe=2; }
    else
    { document.bgColor="aqua"; farbe=1; }
    i = i + 1;
    if(i >= 10)
        window.clearInterval(aktiv);
}
//-->
</script>
</body></html>

```

Das Beispiel definiert mit `setInterval()`, dass die Funktion `Farbe()` alle 1000 Millisekunden, also jede Sekunde einmal aufgerufen wird. Wichtig ist, dass der Aufruf in einer Variablen gespeichert wird, im Beispiel in der Variablen `aktiv`. Diese Variable wird bei der Methode `clearInterval()` benötigt, um den Vorgang zu stoppen. Im Beispiel wechselt die Funktion `Farbe()` bei jedem Aufruf die Hintergrundfarbe der Datei (`document.bgColor`). Gleichzeitig wird ein Zähler hochgezählt. Wenn er größer/gleich 10 ist, wird die Methode `clearInterval()` aufgerufen, die den Endlosvorgang stoppt. Dabei wird die zuvor definierte Variable `aktiv` als Parameter übergeben.




**setTimeout()**

Führt eine Anweisung nach einer bestimmten Verzögerungszeit aus. Erwartet zwei Parameter:

1. *Code* = Eine JavaScript-Anweisung, die wiederholt werden soll. Meistens ein Funktionsaufruf.
2. *Pause* = Wert in Millisekunden bis zum Ausführen.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Hinweis() {
var x = confirm("Sie sind jetzt schon 10 Minuten auf dieser Seite. Fortfahren?");
if(x == false) top.close();
}
window.setTimeout("Hinweis()",600000);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel ruft nach 600000 Millisekunden, also nach 10 Minuten, eine Funktion `Hinweis()` auf, sofern der Anwender nach dieser Zeit immer noch auf der Seite ist. Die Funktion fragt ihn mit der `confirm()`-Methode, ob er noch länger auf der Seite bleiben möchte. Wenn nicht, wird das Hauptfenster geschlossen. Dazu dient der Befehl `top.close()`, der die `close()`-Methode auf die oberste Fenster-Instanz des Browsers anwendet.

**Beachten Sie:** An `setTimeout()` gekoppelte Funktionsaufrufe können auch Parameter enthalten. Berücksichtigen Sie dabei, dass lokale Variablen dann nicht mehr zur Verfügung stehen. Eine Übergabe der Werte mittels Zeichenkettenverknüpfung ist jedoch jederzeit möglich.



**stop()**

Entspricht einem Klick auf den `Stop`-Button im Browser. Der Ladevorgang einer Seite wird dadurch abgebrochen. Das Beispiel definiert einen Verweis, bei dessen Anklicken der Ladevorgang einer Seite abgebrochen wird:

```
<html><head><title>Test</title>
</head><body>
Diese Seite hat einen Umfang von 300 Kilobyte:<br>
<a href="javascript:stop()">abbrechen</a>
</body></html>
```

## 20.3 frames

### Eigenschaft

`length` (Anzahl Frames)

**Unterobjekte**

```
document
event
history
location
```

**frames: Allgemeines zur Verwendung**

Mit dem Objekt `frames` haben Sie Zugriff auf Frames, also auf ein Frameset und seine Frame-Fenster, die in einer HTML-Datei definiert sind. Da jedes Frame-Fenster aus Sicht des Browsers ein eigenes Fenster darstellt, ist das `frames`-Objekt lediglich eine Variante des `window`-Objekts, also des allgemeinen Fenster-Objekts. Alle Eigenschaften und Methoden, die zum `window`-Objekt gehören, lassen sich auch auf das `frames`-Objekt, also auf einzelne Frame-Fenster anwenden. Das `frames`-Objekt stellt nur deshalb ein eigenes Objekt dar, weil es eine zusätzliche Eigenschaft enthält und weil bei der Adressierung von Frame-Fenstern in JavaScript ein paar Besonderheiten zu beachten sind:

Schema 1 / Beispiel 1:

```
parent.frames[#].Eigenschaft/Methode();

parent.frames[1].location.href = "sport.html";
```

Schema 2 / Beispiel 2:

```
parent.NameDesFrames.Eigenschaft/Methode();

parent.RechtesFenster.close();
```

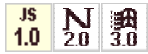
`parent` ist ein reservierter Fenstername, den Sie zum Ansprechen von Fenstern innerhalb eines Framesets verwenden können. Anstelle von `parent` können Sie auch `top` verwenden. Mit `parent` sprechen Sie das Elternfenster des aktuellen Frame-Fensters an, mit `top` in jedem Fall das oberste Anzeigefenster des Browsers.

Frame-Fenster können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1):  
Bei Verwendung von Indexnummern geben Sie `frames` an und dahinter in eckigen Klammern, das wievielte Frame-Fenster im Frameset Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. das erste Frame-Fenster sprechen Sie mit `frames[0]` an, das zweite Frame-Fenster mit `frames[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<frame>`-Befehle im Frameset definiert sind.
- Mit dem Namen des Frame-Fensters (wie in Schema 2 / Beispiel 2):  
Dabei geben Sie den Namen an, der bei der HTML-Definition des Frame-Fensters bei `<frame ...>` im Attribut `name=` definiert worden ist.

Da das `frames`-Objekt eigentlich eine Variante des `window`-Objekts ist, können Sie nicht nur alle Eigenschaften und Methoden des `window`-Objekts auf das `frames`-Objekt anwenden, sondern auch alle anderen, die in der Hierarchie unterhalb des `window`-Objekts stehen. Ebenso, wie Sie beispielsweise mit `window.document.forms[0].elements[0].value = "Stefan"` den Inhalt eines Formular-Eingabefelds in einer gewöhnlichen HTML-Datei ändern können, können Sie

das Gleiche innerhalb eines Framesets etwa mit `parent.frames[1].document.forms[0].elements[0].value = "Stefan"` erreichen. Ein typisches Anwendungsbeispiel für den Einsatz des Frame-Objekts ist das Beispiel *Zwei Frames gleichzeitig ändern* in Kapitel 19.1.



### Auf JavaScripts in anderen Frames zugreifen

Mit der gleichen Syntax, wie Sie auf HTML-Elemente in anderen Frame-Fenstern zugreifen können, können Sie auch auf JavaScripts zugreifen, die in einem anderen Frame-Fenster aktuell notiert sind. So können Sie Variablen des anderen Scripts auslesen und Funktionen des anderen Scripts aufrufen.

Das folgende Beispiel zeigt ein Frameset mit zwei Fenstern oben und unten. Das obere Fenster hat bei der Frameset-Definition den Namen `oberesFenster` erhalten. Aus dem unteren Frame-Fenster wird nun auf ein JavaScript im oberen Frame-Fenster zugegriffen.

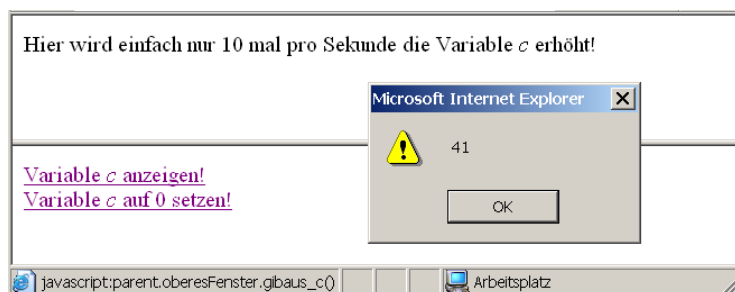
Beispieldatei im oberen Frame-Fenster:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var c = 0;

function gibaus_c() {
  alert(c);
}
function erhoehe_c() {
  c++;
  window.setTimeout("erhoehe_c()",100);
}
//-->
</script>
</head><body onLoad="erhoehe_c()">
<p>Hier wird einfach nur 10 mal pro Sekunde die Variable <i>c</i> erh&ouml;ht!</p>
</body></html>
```

Beispieldatei im unteren Frame-Fenster:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function reset_c() {
  parent.oberesFenster.c = 0;
}
//-->
</script>
</head><body>
<p>
<a href="javascript:parent.oberesFenster.gibaus_c()">Variable <i>c</i>
anzeigen!</a><br>
<a href="javascript:reset_c()">Variable <i>c</i> auf 0 setzen!</a>
</p>
</body></html>
```



20.1: Im unteren Frame greift ein Script auf Variablen aus dem oberen Frame zu.

Im oberen Fenster ist ein Script mit zwei Funktionen namens `gibaus_c()` und `erhoehe_c()` notiert. Die Funktion `erhoehe_c()` wird mit dem Event-Handler `onLoad` im einleitenden `<body>`-Tag erstmals gestartet und ruft sich dann mit Hilfe der Methode `setTimeout()` alle 100 Millisekunden, also 10-mal pro Sekunde selber wieder auf. Dabei erhöht sie jedes Mal die Variable `c` um 1.

Im unteren Fenster sind zwei Verweise notiert. Der erste gibt den aktuellen Zählerstand von `c` aus. Dazu wird mit `parent.oberesFenster.gibaus_c()` auf die Funktion `gibaus_c()` im anderen Frame-Fenster zugegriffen. Der zweite Verweis ruft eine Funktion `reset_c()` auf, die im Dateikopf des eigenen Dokuments notiert ist. Diese greift jedoch mit `parent.oberesFenster.c` auf die Variable `c` im anderen Frame-Fenster zu und setzt diese auf 0 zurück.



Speichert, wie viele Frame-Fenster ein übergeordnetes Fenster enthält. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
for(var i=0; i < parent.frames.length; i++)
    alert(parent.frames[i].name);
//-->
</script>
</head><body>
</body></html>
```

Mit `parent.frames.length` oder `top.frames.length` können Sie die Anzahl der Frame-Fenster ermitteln. Im Beispiel wird angenommen, dass die HTML-Datei innerhalb eines Framesets aufgerufen wird. Beim Einlesen der Datei gibt die Datei mit der Methode `alert()` nacheinander die Namen aller Frame-Fenster im Frameset aus. Dazu dient die `for`-Schleife. Die Eigenschaft `parent.frames.length` dient als Abbruchbedingung für die Schleife. Da der Schleifen-zähler `i` bei jedem Schleifendurchlauf um 1 erhöht wird, wird mit `parent.frames[i]` jeweils das nächste Frame-Fenster angesprochen.



## 20.4 document

### **Eigenschaften**

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <code>alinkColor</code>     | (Farbe für Verweise beim Anklicken)       |
| <code>bgColor</code>        | (Hintergrundfarbe)                        |
| <code>charset</code>        | (verwendeter Zeichensatz)                 |
| <code>cookie</code>         | (beim Anwender speicherbare Zeichenkette) |
| <code>defaultCharset</code> | (normaler Zeichensatz)                    |
| <code>fgColor</code>        | (Farbe für Text)                          |
| <code>lastModified</code>   | (letzte Änderung am Dokument)             |
| <code>linkColor</code>      | (Farbe für Verweise)                      |
| <code>referrer</code>       | (zuvor besuchte Seite)                    |
| <code>title</code>          | (Titel der Datei)                         |
| <code>URL</code>            | (URL-Adresse der Datei)                   |
| <code>vlinkColor</code>     | (Farbe für Verweise zu besuchten Zielen)  |

### **Methoden**

|                                     |                                          |
|-------------------------------------|------------------------------------------|
| <code>captureEvents()</code>        | (Ereignisse überwachen)                  |
| <code>close()</code>                | (schließen)                              |
| <code>createAttribute()</code>      | (Attributknoten erzeugen)                |
| <code>createElement()</code>        | (Elementknoten erzeugen)                 |
| <code>createTextNode()</code>       | (Textknoten erzeugen)                    |
| <code>getElementById()</code>       | (HTML-Elementzugriff über id-Attribut)   |
| <code>getElementsByName()</code>    | (HTML-Elementzugriff über name-Attribut) |
| <code>getElementsByTagName()</code> | (HTML-Elementzugriff über Elementliste)  |
| <code>getSelection()</code>         | (selektierter Text)                      |
| <code>handleEvent()</code>          | (Ereignisse verarbeiten)                 |
| <code>open()</code>                 | (Dokument öffnen)                        |
| <code>releaseEvents()</code>        | (Ereignisse abschließen)                 |
| <code>routeEvent()</code>           | (Event-Handler-Hierarchie durchlaufen)   |
| <code>write()</code>                | (ins Dokumentfenster schreiben)          |
| <code>writeln()</code>              | (zeilenweise schreiben)                  |

### **Unterobjekte**

#### HTML-Elemente

`node`  
`all`  
`anchors`  
`applets`  
`forms`  
`images`  
`layers`  
`links`  
`plugins`

**document: Allgemeines zur Verwendung**

Das `document`-Objekt bezieht sich auf den Inhalt, der in einem Browser-Fenster angezeigt wird. In der Objekthierarchie von JavaScript liegt es unterhalb des `window`-Objekts. Im Document Object Model (DOM) des W3-Konsortiums ist das `document`-Objekt das Ausgangsobjekt für den Elementbaum. Die Elemente eines HTML-Dokuments stellen dem DOM zufolge also Unterobjekte des `document`-Objekts dar. Die einzelnen HTML-Elemente können dabei eigene Eigenschaften und Methoden haben. Diese werden im Abschnitt 20.5 über *HTML-Elementobjekte* beschrieben. Entscheidend für den Zugriff auf den HTML-Elementbaum sind beim `document`-Objekt die DOM-Methoden `getElementById` und `getElementsByName`.

Darüber hinaus enthält das `document`-Objekt selber eine Reihe wichtiger Eigenschaften und Methoden, die dokumentglobale Eigenschaften speichern oder Funktionen ausführen. Die meisten davon werden auch vom DOM unterstützt. Das Ansprechen dieser Eigenschaften und Methoden funktioniert wie folgt:

```
x = document.title;
x = window.document.title;

Zweitfenster = window.open("datei2.htm", "Fenster");
x = Zweitfenster.document.title;

y = parent.frames[2].document.bgColor;

document.write("Hallo Welt");
window.document.write("Hallo Welt");

Zweitfenster.document.close();
parent.frames[2].document.close();
```

Eigenschaften des `document`-Objekts, wie etwa der Titel der angezeigten HTML-Datei (`<title>...</title>`), können Sie mit `document` ansprechen, den Titel der Datei also `document.title`. Entsprechend werden Methoden des `document`-Objekts angesprochen, zum Beispiel die `write`-Methode mit `document.write()`. Sie können auch `window.document.title` oder `window.document.write()` notieren. Da es sich dabei aber um den Inhalt des aktuellen Fensters handelt, ist die Angabe des Fensterobjekts nicht zwingend erforderlich. Anders beim Zugriff auf andere Fenster oder Frame-Fenster. In diesen Fällen müssen Sie das Fenster korrekt adressieren. Lesen Sie für zusätzliche Fenster den Abschnitt 20.2 zur Verwendung des `window`-Objekts und für Frame-Fenster den Abschnitt 20.3 über die Verwendung des `frames`-Objekts.



Speichert die Farbe für Verweise, während diese angeklickt werden, wie beim Attribut `alink` im `<body>`-Tag oder vom Anwender in seinen Browser-Einstellungen festgelegt ist (siehe auch Kapitel 4.2.1 *Farben für Hintergrund, Text und Verweise*). Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var i = 0;
function AndereFarbe() {
```

```

if(i == 0) document.alinkColor = "#000000";
else if(i == 1) document.alinkColor = "#FF0000";
else if(i == 2) document.alinkColor = "#0000FF";
else if(i == 3) document.alinkColor = "green";
i = i + 1; if(i > 3) i = 0;
}
//-->
</script>
</head><body>
<a href="javascript:AndereFarbe()">Verweisfarbe</a>
</body></html>

```

Das Beispiel ruft beim Anklicken des Verweises eine Funktion namens `AndereFarbe()` auf. Diese Funktion ändert bei jedem ihrer Aufrufe den Wert für `document.alinkColor`. Beachten Sie, dass Farben dabei entweder hexadezimal notiert werden oder in Form erlaubter Farbnamen (siehe auch Kapitel 3.5 *Farben definieren in HTML*). Der Wert muss in jedem Fall in Anführungszeichen stehen.

**Beachten Sie:** Diese klassische, im herkömmlichen JavaScript gültige Implementierung von `alinkColor` als direkte Eigenschaft des `document`-Objekts wird vom DOM missbilligt. Laut DOM soll das HTML-Elementobjekt `body` diese Eigenschaft besitzen. Opera 5.12 interpretiert die Eigenschaft `alinkColor` nicht.



Speichert die Hintergrundfarbe der HTML-Datei, wie sie beim Attribut `bgcolor=` im `<body>`-Tag oder vom Anwender in seinen Browser-Einstellungen festlegbar ist (siehe auch Kapitel 4.2.1 *Farben für Hintergrund, Text und Verweise*). Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var X = new Array("0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F");
var x1 = 0, x2 = 0;
function setColor() {
  document.bgColor = "#" + X[x1] + X[x2] + X[x1] + X[x2] + X[x1] + X[x2];
  x2 = x2 + 1;
  if(x2 % 16 == 0) { x2 = 0; x1 = x1 + 1; }
}
for(var i = 0; i < 255; ++i) {
  window.setTimeout("setColor()",20);
}
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel blendet beim Einlesen die Hintergrundfarbe von Schwarz über Graustufen nach Weiß auf. Dazu wird ein Array `X` definiert, in dem die Hexadezimalziffern 0 bis 9 und A bis F gespeichert werden. In einer `for`-Schleife wird von 0 bis 255 gezählt. Der Grund ist, dass 255 der maximale Wert (dezimal gesehen) eines RGB-Farbwerts sein kann. Innerhalb der Schleife wird

mit `window.setTimeout()` nach jeweils 20 Millisekunden Verzögerungszeit die Funktion `setColor()` aufgerufen. Innerhalb dieser Funktion wird der Eigenschaft `document.bgColor` jeweils ein neuer Wert zugewiesen. Dabei wird aus den zuvor definierten Ziffern das Arrays `x` eine gültige Farbangabe konstruiert.

Beachten Sie, dass Farben entweder hexadezimal notiert werden oder in Form erlaubter Farbnamen (siehe auch Kapitel 3.5 *Farben definieren in HTML*). Der Wert muss in jedem Fall in Anführungszeichen stehen. Im obigen Beispiel tut er das nicht, weil es sich um eine Zeichenkette handelt, die dynamisch konstruiert wird. Fixe Angaben wären beispielsweise `"#0099CC"` oder `"blue"`.

**Beachten Sie:** Diese klassische, im herkömmlichen JavaScript gültige Implementierung von `bgColor` als direkte Eigenschaft des `document`-Objekts wird vom DOM missbilligt. Laut DOM soll das HTML-Elementobjekt `body` diese Eigenschaft besitzen. In vielen Dokumentationen wird angegeben, dass die hexadezimale Farbangabe ohne das Gatterzeichen (`#`) zu erfolgen hat. Der Netscape 6 interpretiert jedoch unter Doctype-Angaben wie z.B. `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">` die Farbe nur dann, wenn ein Gatterzeichen (`#`) notiert wurde. Da die Verwendung des Gatterzeichens auch in älteren Browsern keine Probleme verursacht, empfiehlt es sich, dieses stets zu notieren.

Opera 5.12 interpretiert die Eigenschaft `document.bgColor` noch nicht. Netscape 6.1 interpretiert das obige Beispiel (im Gegensatz zu Netscape 4.x) nicht.



Speichert den definierten Zeichensatz eines Dokuments. Das Beispiel enthält im Dateikopf eine Angabe zur Definition des Zeichensatzes. Dort wird `iso-8859-1` definiert (der Standardzeichensatz für westliche Sprachen, inklusive der deutschen). Ferner enthält die Datei einen Verweis. Beim Anklicken wird die Funktion `iso5()` aufgerufen. Diese Funktion ändert den Zeichensatz auf `iso-8859-5` (Kyrillisch). Anschließend werden dynamisch Inhalte in das Dokumentfenster geschrieben. Der MS Internet Explorer schreibt den angegebenen Text in das Dokumentfenster, schaltet dabei jedoch auf eine spezielle Unicode-Schriftart zur Darstellung um. Beachten Sie, dass zum Schreiben des neuen Inhalts die Methoden `open()` und `close()` verwendet werden.

```
<html><head><title>Test</title>
<meta http-equiv="content-type" content="text/html;CHARSET=iso-8859-1">
<script type="text/javascript">
<!--
function iso5() {
  document.charset="iso-8859-5";
  document.open();
  document.write("Text mit ä ö ü");
  document.close();
}
//-->
</script>
</head><body>
<a href="javascript:iso5()">iso-8859-5</a>
</body></html>
```

**Beachten Sie:** Der MS Internet Explorer akzeptiert den geänderten Zeichensatz oft erst nach dem Reload der Seite und einer erneuten Ausführung der Änderung.



Cookies (*cookies* = Kekse – die Herkunft des Namens ist unbekannt) bieten Ihnen die Möglichkeit, direkt aus einer HTML-Datei heraus Daten auf dem Rechner des Anwenders zu speichern und beim erneuten Aufruf der gleichen HTML-Datei wieder auszulesen. So kann eine WWW-Seite dynamisch auf gespeicherte Daten reagieren. Es ist nur möglich, diejenigen Cookies auszulesen, die man selbst gesetzt hat. Eine Virenübertragung mit Cookies ist ausgeschlossen. Ein Cookie ist in etwa das Gleiche wie ein Eintrag in einer *ini*-Datei unter MS-Windows. Es wird eine Variable mit einem zugewiesenen Wert abgespeichert, zum Beispiel Datum und Uhrzeit des letzten Besuchs der WWW-Seite. Es können keine Rechnerdaten des Anwenders ausgelesen werden. Angesichts des vorherrschenden Misstrauens bei Anwendern, die nicht wissen, was ein Cookie ist, sollten Sie Cookies nur verwenden, wenn Sie einen Grund dazu haben.

Netscape verwaltet Cookies in einer Datei namens *cookies.txt* auf dem Rechner des Anwenders. Der MS Internet Explorer verwaltet die Cookies in einem Unterverzeichnis des Windows-Verzeichnisses. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var gespeichert = "nichts gespeichert";
  if(document.cookie)
    gespeichert = document.cookie;
  else
    document.cookie = "Zeitstempel=" + document.lastModified;
  alert(document.lastModified + " - " + gespeichert);
//-->
</script>
</head><body>
</body></html>
```

Das Lesen eines Cookies funktioniert nach dem Schema `x = document.cookie`. Das Schreiben eines Cookies funktioniert nach dem Schema `document.cookie = x`. Das Beispiel fragt beim Einlesen der Datei ab, ob diese Datei schon einmal einen Cookie gesetzt hat. Eine solche Abfrage ist mit `if(document.cookie)` möglich. Im Beispiel wird die Variable `gespeichert` mit dem Wert des gespeicherten Cookies versorgt, falls ein Cookie vorhanden ist. Im `else`-Zweig des Beispiels steht der Fall, dass noch kein Cookie gesetzt wurde. In diesem Fall wird als Cookie die Zeichenkette `Zeitstempel=` und dahinter der Wert von `lastModified` geschrieben. Am Ende werden mit `alert()` zwei Werte ausgegeben: nämlich der Wert von `document.lastModified` und der Wert der Variablen `gespeichert`. Wenn ein Cookie gesetzt ist (also ab dem zweiten Aufruf der Datei), wird der Inhalt des Cookies ausgegeben, der sich von `document.lastModified` dann unterscheidet, wenn die Datei mittlerweile geändert wurde.

Beim Setzen eines Cookies können Sie außerdem einen Verfallszeitpunkt angeben. Wenn der Verfallszeitpunkt erreicht ist, löscht der Browser den Cookie selbstständig. Notieren Sie dazu eine Anweisung der Form: `document.cookie = "expires=" + Verfallsdatum`.

Damit der Browser diese Eigenschaft interpretiert, muss das Verfallsdatum mit der Methode `toGMTString()` des `Date`-Objektes formatiert werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
    var ablauf = new Date();
    var infuenfTagen = ablauf.getTime() + (5 * 24 * 60 * 60 * 1000);
    ablauf.setTime(infuenfTagen);
    document.cookie = "Name=SELFHTML; expires=" + ablauf.toGMTString();
    alert(document.cookie) ;

//-->
</script>
</head><body>
</body></html>
```

In der Variablen `ablauf` wird eine neue Instanz des `Date`-Objekts angelegt. Diese enthält das aktuelle Datum. Das Verfallsdatum ist 5 Tage nach dem Setzen des Cookies. Zur Bestimmung des Verfallsdatums wird das aktuelle Datum mit der Methode `getTime()` in Millisekunden umgewandelt. Zu diesem Wert wird die Anzahl der Millisekunden für 5 Tage addiert. Um den Millisekundenwert zu ermitteln, können Sie wie im Beispiel eine Multiplikationskette übergeben: 1000 Millisekunden sind eine Sekunde, mal 60 macht eine Minute, mal 60 macht eine Stunde, mal 24 einen Tag, und mal 5 macht fünf Tage.

Um eine gültige Datumszeichenkette zu erhalten, wird mit der Methode `setTime()` dem Datum ein neuer Wert zugewiesen. In der Variablen `ablauf` ist dann das korrekte Ablaufdatum gespeichert. Dem Cookie wird schließlich das mit der Methode `toGMTString()` formatierte Datum übergeben.

**Beachten Sie:** Jede HTML-Datei hat Kenntnis von allen Cookies, die von Dateien aus dem gleichen oder einem höheren Verzeichnis gesetzt wurden. Angenommen, bei einer Domain <http://www.test.de> mit dem Unterverzeichnis `/A` gibt es in diesem Unterverzeichnis sowie im Wurzelverzeichnis / jeweils eine Datei namens `x.htm`. Beide Dateien enthalten jeweils ein JavaScript mit der Anweisung:

```
document.cookie = location.href;
```

Damit wird der URI der Datei in einem Cookie gespeichert. Ferner gibt es in beiden Verzeichnissen eine Datei `y.htm`, die jeweils folgende JavaScript-Anweisung enthält:

```
document.write(document.cookie);
```

Damit wird ein vorhandener Cookie dynamisch ins Dokumentfenster geschrieben. Bei der angenommenen Anordnung liefern folgende Dateien folgende Ausgaben:

<http://www.test.de/y.htm> liefert:

<http://www.test.de/x.htm> (kann also den Cookie lesen, den `x.htm` im gleichen Verzeichnis gesetzt hat)

<http://www.test.de/A/y.htm> liefert:

<http://www.test.de/A/x.htm>; <http://www.test.de/x.htm> (kann also den Cookie lesen, den `x.htm` im gleichen Verzeichnis gesetzt hat, sowie den Cookie, den die `x.htm` im Verzeichnis oberhalb gesetzt hat)

Im MS Internet Explorer ab Version 5 und im Netscape 6 können Sie mittels der Eigenschaft `navigator.cookieEnabled` abfragen, ob der Anwender Cookies aktiviert hat. Opera 5.12

erlaubt kein Setzen von Cookies im Protokoll `file`. Mit anderen Worten: Cookies funktionieren nicht auf der lokalen Festplatte, sondern nur beim Ausführen von einem Web-Server.



Speichert den Zeichensatz, den der Anwender als Standard im Browser eingestellt hat. Wenn die Datei keine Meta-Angabe zum verwendeten Zeichensatz enthält, wird der Standardzeichensatz angenommen. Das Beispiel gibt beim Einlesen der Datei mit `alert()` aus, welchen Zeichensatz der Anwender als Standardzeichensatz eingestellt hat:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
alert("Sie haben als Zeichensatz eingestellt: " + document.defaultCharset);
//-->
</script>
</head><body>
</body></html>
```



Speichert die Farbe für Text, wie beim Attribut `text=` im `<body>`-Tag oder vom Anwender in seinen Browser-Einstellungen festlegbar (siehe auch Kapitel 4.2.1 *Farben für Hintergrund, Text und Verweise*). Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(parent.frames[0].document.fgColor == "#000000")
  document.fgColor = "#FF0000";
//-->
</script>
</head><body>
Ein kleiner Text
</body></html>
```

Das Beispiel nimmt an, dass sich die aktuelle Datei innerhalb eines Framesets befindet. Beim Einlesen der Datei wird geprüft, ob die Textfarbe eines anderen (des ersten) Frame-Fensters schwarz ist (`if(parent.frames[0].document.fgColor == "#000000")`). Wenn dies der Fall ist, wird die Textfarbe der aktuellen Datei auf rot gesetzt (`document.fgColor = "#FF0000";`). Beachten Sie, dass Farbänderungen für die Textfarbe in Netscape bis einschließlich Version 4 nur Wirkung haben, **bevor** irgendwelcher HTML-Text eingelesen wurde. Ein gegebenenfalls im `<body>`-Tag gesetztes Attribut überschreibt die Farbzuzuweisung, falls diese zuvor im Dateikopf mit JavaScript gesetzt wurde. Beachten Sie bei den Farbangaben, dass Farben entweder hexadezimal notiert werden oder in Form erlaubter Farbnamen (siehe auch Kapitel 3.5, *Farben definieren in HTML*). Der Wert muss in jedem Fall in Anführungszeichen stehen.

**Beachten Sie:** Diese klassische, im herkömmlichen JavaScript gültige Implementierung von `alinkColor` als direkte Eigenschaft des `document`-Objekts wird vom DOM missbilligt. Laut DOM soll das HTML-Elementobjekt `body` diese Eigenschaft besitzen. Opera 5.12 interpretiert diese Eigenschaft noch nicht.

**lastModified**

Speichert Datum und Uhrzeit der letzten Änderung der Datei, und zwar im internationalen Format nach GMT (Greenwich-Zeit). Das Beispiel schreibt mit der Methode `write()` den Zeitpunkt der letzten Änderung dynamisch in die HTML-Datei:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write("letzter Update: " + document.lastModified);
//-->
</script>
</body></html>
```

**Beachten Sie:** Die Formatierung der Zeichenkette ist stark abhängig vom Browser und dem verwendeten Betriebssystem. Berücksichtigen Sie insbesondere bei der Weiterverarbeitung dieser Eigenschaft mit dem `Date`-Objekt, dass einige Versionen des Netscape 4 eine zweistellige Jahreszahl an das `Date`-Objekt übergeben. Dies führt unter anderem dazu, dass ein falscher Wochentag und ein falsches Jahr zurückgegeben werden.

**linkColor**

Speichert die Farbe für Verweise, wie beim Attribut `link=` im `<body>`-Tag oder vom Anwender in seinen Browser-Einstellungen festlegbar (siehe auch Kapitel 4.2.1 *Farben für Hintergrund, Text und Verweise*). Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
document.linkColor = parent.frames[0].document.linkColor;
//-->
</script>
</head><body>
<a href="#">Ein Verweis</a>
</body></html>
```

Das Beispiel nimmt an, dass sich die aktuelle Datei innerhalb eines Framesets befindet. Beim Einlesen der Datei wird die Verweisfarbe der aktuellen Datei auf die gleiche Farbe gesetzt wie die Verweisfarbe der Datei, die sich in einem anderen (dem ersten) Frame-Fenster befindet.

Beachten Sie, dass Farbänderungen für die Verweisfarbe in Netscape bis einschließlich Version 4 nur Wirkung haben, **bevor** irgendwelcher HTML-Text eingelesen wurde. Ein gegebenenfalls im `<body>`-Tag gesetztes Attribut überschreibt die Farbzuzuweisung, falls diese zuvor im Dateikopf mit JavaScript gesetzt wurde. Beachten Sie bei den Farbangaben, dass Farben entweder hexadezimal notiert werden oder in Form erlaubter Farbnamen (siehe auch Kapitel 3.5 *Farben definieren in HTML*). Der Wert muss in jedem Fall in Anführungszeichen stehen.

**Beachten Sie:** Diese klassische, im herkömmlichen JavaScript gültige Implementierung von `alinkColor` als direkte Eigenschaft des `document`-Objekts wird vom DOM missbilligt. Laut DOM soll das HTML-Elementobjekt `body` diese Eigenschaft besitzen. Opera 5.12 interpretiert diese Eigenschaft noch nicht.





Speichert den URI, von dem aus die aktuelle Datei aufgerufen wurde. Voraussetzung ist, dass er die aktuelle Datei über einen Verweis aufgerufen hat. Wenn er die Adresse der Datei stattdessen etwa als Lesezeichen ausgewählt oder direkt in die Adresszeile des Browsers eingegeben hat, ist in `document.referrer` nichts gespeichert. Beim MS Internet Explorer und in Netscape 6 wird in `document.referrer` nur dann etwas gespeichert, wenn der Aufruf der aktuellen Datei über das HTTP-Protokoll erfolgte, also mit einem Verweis vom Typ `http://...` Das Beispiel gibt im HTML-Text mit der Methode `write()` aus, von woher der Anwender die aktuelle Datei aufgerufen hat:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  document.write("<h1>Lieber Besucher</h1>");
  document.write("Du hast also von " + document.referrer + " hierher gefunden");
//-->
</script>
</body></html>
```

**Beachten Sie:** Opera 5.12 interpretiert diese Eigenschaft noch nicht.



Speichert den Titel einer HTML-Datei, wie er bei `<title>...</title>` vergeben wurde. Das Beispiel schreibt mit der Methode `write()` den Titel der HTML-Datei dynamisch als Inhalt in eine Überschrift 1. Ordnung (`<h1>...</h1>`):

```
<html><head><title>Test</title>
</head><body>
<h1>
<script type="text/javascript">
<!--
  document.write(document.title);
//-->
</script>
</h1>
</body></html>
```

**Beachten Sie:** Im MS Internet Explorer und im Netscape Navigator können Sie diese Eigenschaft auch ändern.



Speichert den vollständigen URI des aktuellen Dokuments. Das Beispiel schreibt mit der Methode `write()` die eigene vollständige Adresse in den Text. So etwas kann durchaus Sinn machen, beispielsweise, wenn die Datei Teil eines Framesets ist. Anwender können sich dann auf die genaue Adresse beziehen, wenn Sie zu dem angezeigten Frame-Inhalt etwas mitteilen wollen.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write("Diese Datei: " + document.URL);
//-->
</script>
</body></html>
```



Speichert die Farbe für Verweise zu bereits besuchten Seiten, wie beim Attribut `vlink=` im `<body>`-Tag oder vom Anwender in seinen Browser-Einstellungen festlegbar (siehe auch 4.2.1 *Farben für Hintergrund, Text und Verweise*). Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
document.vlinkColor = parent.frames[0].document.vlinkColor;
//-->
</script>
</head><body>
<a href="../document.htm">Verweis</a>
</body></html>
```

Das Beispiel nimmt an, dass sich die aktuelle Datei innerhalb eines Framesets befindet. Beim Einlesen der Datei wird die Farbe für Verweise zu besuchten Seiten in der aktuellen Datei auf die gleiche Farbe gesetzt wie bei der Datei, die sich in einem anderen (dem ersten) Frame-Fenster befindet. Beachten Sie, dass Farbänderungen für die Verweisfarbe in Netscape bis einschließlich Version 4 nur Wirkung haben, **bevor** irgendwelcher HTML-Text eingelesen wurde. Ein gegebenenfalls im `<body>`-Tag gesetztes Attribut überschreibt die Farbzuzuweisung, falls diese zuvor im Dateikopf mit JavaScript gesetzt wurde. Beachten Sie bei den Farbangaben, dass Farben entweder hexadezimal notiert werden oder in Form erlaubter Farbnamen (siehe auch Kapitel 3.5 *Farben definieren in HTML*). Der Wert muss in jedem Fall in Anführungszeichen stehen.

**Beachten Sie:** Diese klassische, im herkömmlichen JavaScript gültige Implementierung von `alinkColor` als direkte Eigenschaft des `document`-Objekts wird vom DOM missbilligt. Laut DOM soll das HTML-Elementobjekt `body` diese Eigenschaft besitzen. Opera 5.12 interpretiert diese Eigenschaft noch nicht.



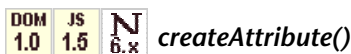
Überwacht Anwenderereignisse im aktuellen Dokument. Funktioniert genau so wie `captureEvents()` beim `window`-Objekt (nähere Informationen siehe dort). Der einzige Unterschied ist, dass Sie mit `document.captureEvents()` nur Ereignisse innerhalb des Dokumentfensters überwachen können, jedoch keine Ereignisse im gesamten Fensterbereich (zu dem beispielsweise auch Titelleiste, Menüleisten usw. gehören).



Schließt einen Dokumentinhalt, der mit der `open()`-Methode geöffnet und mit den Methoden `write()` oder `writeln()` beschrieben wurde. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.open();
document.write(document.URL);
document.close();
//-->
</script>
</body></html>
```

Das Beispiel öffnet den Dokumentinhalt zum Schreiben, dann schreibt es etwas (den aktuellen URI) mit der Methode `write()`, und zuletzt schließt es den Vorgang mit `document.close()` ab. Mit dem Schließen des Dokumentes wird dem Browser signalisiert, dass der Aufbau der Seite abgeschlossen ist. Dies ist besonders beim dynamischen Schreiben von Frames und zum Beschreiben von Layern erforderlich.



Erzeugt ein neues Attribut für den Elementbaum. lässt sich auf HTML-, aber auch auf XML-Dokumente anwenden. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<h1>Eine sehr dynamische Seite</h1>
<script type="text/javascript">
<!--
var Ausrichtung = document.createAttribute("align");
Ausrichtung.nodeValue = "right";
var Element = document.getElementsByTagName("h1")[0];
Element.setAttributeNode(Ausrichtung);
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Überschrift erster Ordnung ohne weitere Attribute. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.createAttribute("align")` ein neues Attribut mit Namen `align` erzeugt, das in diversen HTML-Elementen zum Ausrichten des Inhalts verwendet werden kann. Das Attribut hat damit aber noch keine Wirkung – es hängt weder im Elementbaum des Dokuments, noch hat es einen Wert. Das Attribut ist lediglich als Objekt erzeugt.

Die Methode `createElement()` gibt das erzeugte Objekt, also in dem Fall das `align`-Attribut, zurück. Der Rückgabewert wird in der Variablen `Ausrichtung` gespeichert. Über diese Variable ist das neu erzeugte Attribut von da ab ansprechbar.

Damit das Attribut einen Inhalt erhält, wird dem Attributknoten über die Objekteigenschaft `nodeValue` der gewünschte Wert `right` zugewiesen. Anschließend müssen das Attribut und sein Wert noch in den vorhandenen Elementbaum eingehängt werden. Dies geschieht in der Anweisung `Element.setAttributeNode(Ausrichtung);` (siehe auch `setAttributeNode()`). Dabei wurde zuvor in der Variablen `Element` das `h1`-Element gespeichert, auf das mit `getElementsByTagName()` zugegriffen wurde. Somit wird die Überschrift dynamisch rechtsbündig ausgerichtet.

**Beachten Sie:** Mit dem Internet Explorer 5.5 war dieses Beispiel nicht nachvollziehbar, wenngleich Microsoft behauptet, diese Methode im Internet Explorer implementiert zu haben. Im MS Internet Explorer 6.0 ist das Beispiel dagegen nachvollziehbar.

#### ***createElement()***

Erzeugt ein neues Element für den Elementbaum. lässt sich auf HTML-, aber auch auf XML-Dokumente anwenden. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<div id="Bereich" style="border:1px black solid; padding:10px">
</div>
<script type="text/javascript">
<!--
  var myH1 = document.createElement("h1");
  var myText = document.createTextNode("Eine sehr dynamische Seite");
  myH1.appendChild(myText);
  var Ausgabebereich = document.getElementById("Bereich");
  Ausgabebereich.appendChild(myH1);
//-->
</script>
</body></html>
```

Das Beispiel enthält einen `div`-Bereich, der zunächst leer ist. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.createElement("h1")` ein neues Element vom Typ `h1` erzeugt, also eine HTML-Überschrift erster Ordnung. Die Überschrift wird damit aber noch nicht angezeigt – sie hängt noch nicht einmal im Elementbaum des Dokuments. Das Element ist lediglich als Objekt erzeugt.

Die Methode `createElement()` gibt das erzeugte Objekt, also in dem Fall das `h1`-Element zurück. Der Rückgabewert wird in der Variablen `myH1` gespeichert. Über diese Variable ist das neu erzeugte Element von da ab ansprechbar.

Damit die Überschrift einen Inhalt erhält, wird mit `createTextNode()` ein Textknoten erzeugt. Mit `appendChild()` wird dann der Textknoten als Kindelement der Überschrift erster Ordnung definiert. Anschließend muss noch die ganze Überschrift in den vorhandenen Elementbaum eingehängt werden. Dies geschieht in der Anweisung `Ausgabebereich.appendChild(myH1);`. Dabei wurde zuvor in der Variablen `Ausgabebereich` das leere `div`-Element gespeichert, auf das mit `getElementById()` zugegriffen wurde. Somit wird die Überschrift und ihr Text also als Kindelement dynamisch in den `div`-Bereich eingefügt.


***createTextNode()***

Erzeugt einen neuen Textknoten für den Elementbaum. Lässt sich auf HTML-, aber auch auf XML-Dokumente anwenden. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p id="Absatz"></p>
<script type="text/javascript">
<!--
  var Zeitstempeltext = document.createTextNode(document.lastModified);
  var TextZuvor = document.createTextNode("Datum des letzten Updates: ");
  document.getElementById("Absatz").appendChild(TextZuvor);
  document.getElementById("Absatz").appendChild(Zeitstempeltext);
//-->
</script>
</body></html>
```

Das Beispiel enthält einen p-Absatz, der zunächst leer ist. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort werden mit `document.createTextNode()` zwei neue Zeichenketten erzeugt, einmal der gespeicherte Wert von `lastModified` und einmal ein statischer Text. Die Texte werden damit aber noch nicht angezeigt – sie sind auch noch nicht in den Elementbaum des Dokuments eingehängt. Die Textknoten sind lediglich als Objekt erzeugt.

Die Methode `createTextNode()` gibt das erzeugte Objekt zurück. Der Rückgabewert wird in den beiden Beispielaufrufen in den Variablen `Zeitstempeltext` und `TextZuvor` gespeichert. Über diese Variablen sind die neu erzeugten Textknoten von da ab ansprechbar.

Mit `appendChild()` werden dann die Textknoten als Kindelemente des über dem JavaScript notierten Textabsatzes definiert. Dazu wird mit `getElementById()` auf das p-Element zugegriffen. Der leere Absatz erhält somit dynamisch einen Inhalt.


***getElementById()***

Greift entsprechend der HTML-Variante des DOM auf ein HTML-Element zu, das ein `id`-Attribut besitzt. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.normal { font-style:normal }
-->
</style>
<script type="text/javascript">
<!--
  function Ausrichten(wie) {
    document.getElementById("unentschlossen").align = wie;
  }
//-->
</script>
</head><body>
```

```
<h1 id="unentschlossen">Wo gehöre ich eigentlich hin?</h1>
<a href="javascript:Ausrichten('left')">links?</a><br>
<a href="javascript:Ausrichten('center')">zentriert?</a><br>
<a href="javascript:Ausrichten('right')">rechts?</a><br>
</body></html>
```

Das Beispiel enthält eine Überschrift erster Ordnung mit dem Attribut `id="unentschlossen"`. Unterhalb davon sind drei Verweise notiert. Jeder der Verweise ruft die Funktion `Ausrichten()` auf, die im Dateikopf notiert ist, und übergibt ihr jeweils einen anderen Wert. Innerhalb der Funktion `Ausrichten()` wird mit `document.getElementById("unentschlossen")` auf die Überschrift erster Ordnung zugegriffen. Die Funktion weist der Eigenschaft `align` den jeweils übergebenen Wert zu. Dadurch wird die Ausrichtung der Überschrift dynamisch geändert.

**Beachten Sie:** Die Methode `getElementById()` regelt lediglich den Zugriff auf HTML-Elemente. Um zu wissen, was Sie mit welchen HTML-Elementen überhaupt tun können, können Sie sich im Kapitel über die HTML-Elementobjekte einen Überblick verschaffen. Weitere Möglichkeiten, um HTML-Elemente dynamisch zu beeinflussen, bietet das `node-Objekt` an.

Sie können an diese Methode auch Variablen übergeben. Dazu notieren Sie einfach den Variablennamen ohne Anführungszeichen anstatt eines festen Wertes, z.B. `getElementById(Variablenname)`. Voraussetzung ist jedoch, dass die Variable auch existiert.

In Opera 5.12 kennt die Methode `getElementById()` zum Ansprechen von Elementen ebenfalls. Das obige Beispiel funktioniert jedoch in diesem Browser nicht.



### `getElementsByName()`

Greift entsprechend der HTML-Variante des DOM auf ein HTML-Element zu, das ein `name`-Attribut besitzt. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.normal { font-style:normal }
-->
</style>
<script type="text/javascript">
<!--
function Ankreuzen() {
    document.getElementsByName("Zutat")[0].checked = true;
}
//-->
</script>
</head><body>
<form name="Formular" action="">
<input type="checkbox" name="Zutat" value="Salami"> Salami <br>
<input type="checkbox" name="Zutat" value="Pilze"> Pilze <br>
<input type="checkbox" name="Zutat" value="Oliven"> Oliven <br>
<input type="button" value="Test" onClick="Ankreuzen()">
</form>
</body></html>
```

Im Beispiel wird eine Gruppe von Checkboxes definiert, die alle den gleichen Namen, nämlich `Zutat` haben. Darunter ist ein Klick-Button notiert, bei dessen Anklicken die Funktion `Ankreuzen()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion kreuzt die erste der drei Checkboxes an. Durch `getElementsByName("Zutat")[0]` greift sie auf das erste Element mit `name="Zutat"` zu. Auch wenn ein solcher Elementname nur einmal im Dokument vorkommt, muss stets mit Array-Syntax auf die Elemente zugegriffen werden. So wäre im Beispiel das Formularelement etwa über `getElementsByName("Formular")[0]` ansprechbar. Die zweite Checkbox mit dem Namen `"Zutat"` wäre über `getElementsByName("Zutat")[1]` ansprechbar usw.

**Beachten Sie:** Die Methode `getElementsByName()` regelt lediglich den Zugriff auf HTML-Elemente. Um zu wissen, was Sie mit welchen HTML-Elementen überhaupt tun können, können Sie sich im Kapitel über die HTML-Elementobjekte einen Überblick verschaffen. Weitere Möglichkeiten, um HTML-Elemente dynamisch zu beeinflussen, bietet das `node`-Objekt an. Sie können an diese Methode auch Variablen übergeben. Dazu notieren Sie einfach den Variablennamen ohne Anführungszeichen anstatt eines festen Wertes, z.B. `getElementsByName(Variablenname)`. Voraussetzung ist jedoch, dass die Variable auch existiert. Opera 5.12 kennt diese Methode zum Ansprechen von Elementen ebenfalls.



### `getElementsByTagName()`

Greift entsprechend der XML-Variante des DOM auf ein beliebiges Element im Elementbaum des Dokuments zu. Maßgeblich ist dabei der Elementname, in HTML also beispielsweise Namen wie `h1`, `blockquote` oder `img`. Für jeden HTML-Elementtyp wird im Dokument ein Array erzeugt, über den die einzelnen Elemente dann ansprechbar sind.

Leider ist diese Methode vom DOM nicht in der HTML-Variante vorgesehen, doch mit den JavaScript-Interpretern der neueren Browser funktioniert sie trotzdem. Deshalb wird sie hier – anders als die übrigen Eigenschaften und Methoden des `document`-Objekt in der XML-Variante des DOM – beschrieben. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.normal { font-style:normal }
-->
</style>
<script type="text/javascript">
<!--
function neueTexte() {
document.getElementsByTagName("p")[0].firstChild.data = "neuer erster";
document.getElementsByTagName("p")[1].firstChild.data = "neuer zweiter";
document.getElementsByTagName("p")[2].firstChild.data = "neuer dritter";
document.getElementsByTagName("h1")[0].firstChild.data = "Alles anders";
}
//-->
</script>
</head><body>
<h1>Dynamisch, sehr dynamisch</h1>
<p>Ein erster Absatz</p>
```

```
<p>Ein zweiter Absatz</p>
<p>Ein dritter Absatz</p>

<p><a href="javascript:neueTexte()">neue Texte</a></p>
</body></html>
```

Im Beispiel werden eine Überschrift erster Ordnung und drei Textabsätze notiert. Keines dieser Elemente enthält irgendein Attribut, über das es individuell angesprochen werden könnte. Unterhalb davon ist ein Verweis notiert, bei dessen Anklicken die Funktion `neueTexte()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion ändert dynamisch die Texte aller drei Textabsätze und der Überschrift.

Über `document.getElementsByTagName("p")[0]` greift sie auf den ersten Textabsatz des Dokuments zu, über `document.getElementsByTagName("p")[1]` auf den zweiten, über `document.getElementsByTagName("h1")[0]` auf die erste Überschrift erster Ordnung usw.

**Beachten Sie:** Die Methode `getElementsByTagName()` regelt lediglich den Zugriff auf Elemente. Um zu wissen, was Sie mit welchen HTML-Elementen überhaupt tun können, können Sie sich im Kapitel über die HTML-Elementobjekte einen Überblick verschaffen. Weitere Möglichkeiten, um HTML-Elemente dynamisch zu beeinflussen, bietet das `node`-Objekt an. Die Eigenschaft `firstChild.data`, die im obigen Beispiel verwendet wird, um den im Element enthaltenen Text dynamisch zu ändern, gehört ebenfalls zum Komplex des `node`-Objekts. Opera Version 5.12 kennt die Methode `getElementsByTagName()` zum Ansprechen von Elementen. Das obige Beispiel funktioniert jedoch in diesem Browser nicht.

### **getSelection()**

Ermittelt Text, der vom Anwender im Dokument selektiert wurde. Im Netscape 4 war diese Methode an das `document`-Objekt gebunden. Seit Netscape 6 ist sie dem `window`-Objekt zugeordnet. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  function selektierterText()
  {
    if (window.getSelection) alert(window.getSelection());
    else if (document.getSelection) alert(document.getSelection());
    else if (document.selection) alert(document.selection.createRange().text);
  }
-->
</script>
</head><body>
viel Text zum Selektieren viel Text zum Selektieren
<form>
<input type="button" value="zeigen" onMouseDown="selektierterText();">
</form>
</body></html>
```

Das Beispiel ruft beim Klick auf den definierten Button die Funktion `selektierterText()` auf. Innerhalb dieser Funktion wird geprüft, ob der Browser die Methode `window.getSelection()`



kennt. Tritt dieser Fall ein, so wird sie aufgerufen und der selektierte Text wird ausgegeben. Kennt der Browser diese Methode nicht, so wird geprüft, ob er die Methode `document.getSelection()` kennt, und gegebenenfalls der selektierte Text ausgegeben.

Der zweite else-if-Zweig des Beispiels behandelt einen Sonderfall für den MS Internet Explorer. Dieser kennt die Methode `getSelection()` nicht, jedoch ein `selection`-Objekt. Mit `document.selection.createRange().text` erhalten Sie hier den selektierten Text. Voraussetzung ist, dass zuvor Text selektiert wurde. Andernfalls liefert die Funktion `selektierterText()` eine leere Zeichenkette ("") als Ergebnis.

### **handleEvent()**

Übergibt ein Ereignis an ein Element, das in der Lage ist, auf das Ereignis zu reagieren. Funktioniert genau so wie `handleEvent()` beim `window`-Objekt (nähere Informationen siehe dort). Der einzige Unterschied ist, dass `document.handleEvent()` nur Ereignisse übergeben kann, die mit `document.captureEvents()` überwacht werden.

### **open()**

Öffnet ein Dokument zum Schreiben. Dabei wird kein Fenster geöffnet, sondern der Fensterinhalt zum Neubeschreiben freigegeben. Falls das Dokument vorher einen Inhalt hatte, zum Beispiel eine zunächst geladene HTML-Datei, sollten Sie zuerst die `close()`-Methode aufrufen.

Kann ohne, mit einem oder mit zwei Parametern aufgerufen werden. Folgende Parameter sind möglich:

1. *Mime-Type* = Eine Bezeichnung des Mime-Typen für die Art der Daten, die in das Dokumentfenster geschrieben werden sollen. So können Sie das Dokumentfenster beispielsweise durch Angabe von "x-world/x-vrml" zum Schreiben von VRML-Code öffnen. Mit `write()`- und `writeln()`-Befehlen können Sie dann dynamisch VRML-Code schreiben.
2. *replace* = mit `document.open("text/html","replace")` öffnen Sie das Dokument zum Schreiben von HTML und bewirken, dass der neue Dokumentinhalt die gleiche Stelle in der History der besuchten Seiten einnimmt wie das Dokument, in dem der `open()`-Befehl steht.

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
function Wechsel() {
  document.open();
  document.write("<a href=\"datei.htm\">Und jetzt steh ich hier</a>");
  document.close();
}
document.open();
document.write("<a href=\"javascript:Wechsel()\">Noch steh ich hier</a>");
document.close();
//-->
</script>
</body></html>
```

Das Beispielscript führt zunächst den unteren Teil des Codes aus, da der obere in die Funktion `Wechsel()` eingebunden ist, die erst bei Aufruf ausgeführt wird. Im unteren Teil wird mit JavaScript dynamisch ein Verweis in das Dokumentfenster geschrieben. Wenn der Anwender den Verweis anklickt, wird die Funktion `Wechsel()` aufgerufen. Diese Funktion öffnet das Dokumentfenster zum neuen Schreiben und schreibt dynamisch einen Verweis auf die aktuelle Datei (im Beispiel wird angenommen, dass diese Datei *datei.htm* heißt). Klickt der Anwender auf den Verweis, wird die Datei erneut geladen, und dadurch wird wieder der erste Verweis geschrieben.

**Beachten Sie:** Das Beispiel funktioniert erst ab Netscape 3.x so wie beschrieben. Netscape 2.x kennt die `document.open()` zwar auch schon, doch leert dabei das Dokumentfenster nicht. Opera 5.12 kennt laut Dokumentation die Methode `open()` und wendet sie in vielen Fällen richtig an. Das vorliegende Beispiel wird jedoch vom Browser nicht abgeschlossen und nur ohne Verwendung von `document.open()` richtig ausgeführt. Als Alternative können Sie die Funktion mit `<a href="#" onclick="Wechsel();return false">` aufrufen.

### **releaseEvents()**

Beendet die Überwachung von Anwenderereignissen. Funktioniert genau so wie `releaseEvents()` beim `window`-Objekt (nähere Informationen siehe dort). Der einzige Unterschied ist, dass `document.releaseEvent()` nur Ereignisüberwachungen beenden kann, die mit `document.captureEvents()` gestartet wurden.

**Beachten Sie:** Netscape 6 interpretiert diese Methode nicht (mehr).

### **routeEvent()**

Übergibt ein Anwenderereignis an das `document`-Objekt. Von dort aus wird das Ereignis immer weiter nach unten durchgereicht, bis sich ein Event-Handler findet, der das Ereignis verarbeitet. Funktioniert genau so wie `routeEvent()` beim `window`-Objekt (nähere Informationen siehe dort). Der einzige Unterschied ist, dass `document.routeEvent()` nicht auf Fenster-Ebene ansetzt, sondern auf Dokumentebene.

**Beachten Sie:** Netscape 6 interpretiert diese Methode nicht (mehr).

### **write()**

Gibt Text im Dokument aus. Erwartet eine Zeichenkette oder einen JavaScript-Ausdruck als Parameter. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
DeinName = prompt("Bitte Deinen Namen angeben:", "Name");
document.write("<b>Hallo " + DeinName + "</b>");
//-->
</script>
</body></html>
```

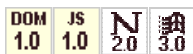
Das Beispiel zeigt beim Einlesen der Datei mit `prompt()` ein Dialogfenster an, in dem der Anwender seinen Namen eingeben soll. Der eingegebene Wert wird in der Variablen `DeinName` gespeichert. Anschließend wird mit `document.write()` das Wort »Hallo« mit einem Leerzeichen dahinter geschrieben. An die Zeichenkette angehängt wird die Variable `DeinName`. Dadurch steht hinterher beispielsweise »Hallo Susi« im Text, wenn in dem Dialogfenster »Susi« als Name angegeben wurde. Damit die Ausgabe in Fettschrift erfolgt, wird sie in das entsprechende HTML-Element gesetzt.

Es ist auch erlaubt, mehrere Zeichenketten oder Ausdrücke durch Kommata getrennt innerhalb von `document.write()` zu verwenden, zum Beispiel:

```
document.write("Sie benutzen ", document.layers ? "Netscape ab V4" : "nicht  
Netscape ab V4");
```

Dieses Beispiel enthält zunächst eine Zeichenkette und dann, durch ein Komma getrennt, einen komplexen Ausdruck, in dem ermittelt wird, ob der Anwender einen Browser benutzt, der das `layers`-Objekt kennt.

**Beachten Sie:** Der Schrägstrich / in schließenden HTML-Tags muss bei der Ausgabe mit dem Zeichen `\` maskiert werden. Das gilt besonders dann, wenn Sie dynamisch weiteren JavaScript-Code in der Form `document.write("<script>...</script>")` schreiben möchten. Maskiert `<script>...</script>` geschrieben, wird der dynamisch geschriebene Script-Bereich als eigenständiger Block interpretiert, und das aktuelle Script wird unabhängig davon weiter ausgeführt.



### *writeln()*

Wie `write()` – fügt jedoch am Ende automatisch ein Zeilenumbruchzeichen ein. Dadurch ist diese Methode vor allem zum Schreiben von Inhalten interessant, bei denen Zeilenumbrüche wichtig sind. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write("<pre>kein Umbruch ");
document.writeln("erste Zeile");
document.writeln("zweite Zeile <\npre>");
document.writeln("dritte Zeile");
document.writeln("vierte Zeile");
//-->
</script>
</body></html>
```

In der ersten Zeile wird mit `document.write()` ein präformatierter Text eingeleitet. Dieser dient im Beispiel dazu, die Zeilenumbrüche sichtbar zu machen. Die Methode `document.write()` erzeugt keinen Zeilenumbruch. Der Inhalt der nachfolgenden Zeile (»erste Zeile«) wird deshalb in der gleichen Zeile ausgegeben. Da diese Zeichenkette jedoch mit `document.writeln()` ausgegeben wird, entsteht im Quelltext am Ende ein Zeilenumbruch. Die nächste Zeile (»zweite Zeile«) wird deshalb in einer neuen Zeile ausgegeben. Zugleich wird im Beispiel in dieser Zeile der präformatierte Abschnitt beendet. Die dritte und die vierte Zeile werden ebenfalls mit `document.writeln()` geschrieben. Intern entstehen Zeilenumbrüche im Quelltext. Bei der Ausgabe

im Browser sind diese jedoch nur als Leerzeichens sichtbar, getreu der HTML-Regel, dass Zeilenumbrüche in Leerzeichen umgewandelt werden.

## 20.5 HTML-Elementobjekte

### *HTML-Elementobjekte: Allgemeines zur Verwendung*

Der HTML-Variante des Document Object Model (DOM) zufolge stellt jedes HTML-Element in einer HTML-Datei ein Objekt dar. Wichtig ist dabei zu wissen, wie mit einer Script-Sprache wie JavaScript auf ein solches HTML-Elementobjekt zugegriffen werden kann. Nach Syntax der HTML-Variante des DOM kann nur auf solche HTML-Elemente zugegriffen werden, die in ihrem einleitenden Tag entweder ein `name`-Attribut oder ein `id`-Attribut enthalten. Nach dem erweiterten XML-DOM lässt sich jedoch auch auf jedes beliebige HTML-Element zugreifen, auch wenn dieses kein `name`- oder `id`-Attribut hat.

- **Zugriff über `name`-Attribut**

Das `name`-Attribut ist in HTML nur bei bestimmten Elementen erlaubt, etwa bei Formular-elementen. Deshalb ist diese Zugriffsmethode auch nur bei den folgenden Elementen möglich, da diese ein `name`-Attribut erlauben:

`a`, `applet`, `form`, `frame`, `img`, `input`, `iframe`, `map`, `meta`, `object`, `param`, `select`, `textarea`

Mit der Methode `document.getElementsByName()` können Sie HTML-Elemente ansprechen, die ein `name`-Attribut haben.

- **Zugriff über `id`-Attribut**

Das `id`-Attribut ist im Gegensatz zum `name`-Attribut in fast **allen** HTML-Elementen erlaubt. Außerdem sollte seine Wertzuweisung ein dokumentweit eindeutiger Name für das Element sein. Dadurch eignet sich diese Zugriffsmethode in den meisten Fällen in der Praxis besser als die über das `name`-Attribut. Mit der Methode `document.getElementById()` können Sie HTML-Elemente ansprechen, die ein `id`-Attribut haben.

- **Zugriff über den Elementbaum**

Wenn Sie auf HTML-Elemente zugreifen möchten, bei denen weder ein `name`-Attribut noch ein `id`-Attribut notiert ist, steht Ihnen ein dritter Weg offen: der Zugriff über den Elementbaum. Dabei können Sie dann beispielsweise auf die »dritte Tabellenzelle in der zweiten Tabellenreihe der vierten Tabelle im Dokument« zugreifen. Mit der Methode `document.getElementsByTagName()` sind solche Zugriffe möglich.

- **Eigenschaften und Methoden von HTML-Elementobjekten**

Jedes HTML-Element hat Eigenschaften. Und zwar stellt jedes erlaubte Attribut eines HTML-Elements eine DOM-Eigenschaft dieses Elements dar. So hat beispielsweise das HTML-Element `input` ein erlaubtes Attribut `value`, und das HTML-Element `h1` hat ein erlaubtes Attribut `align`. Im DOM gibt es demnach also ein `input`-Elementobjekt mit der Eigenschaft `value`, und ein `h1`-Elementobjekt mit der Eigenschaft `align`.

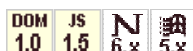
Darüber hinaus definiert das DOM für einige der HTML-Elemente auch Methoden. So gibt es für das `form`-Elementobjekt (also das DOM-Objekt des HTML-Elements `form`) die Methoden `submit()` (Formular absenden) und `reset()` (Formulareingaben verwerfen).

Für alle Eigenschaften und Methoden von HTML-Elementobjekten gelten die zuvor genannten drei Zugriffsmethoden. Wenn Sie beispielsweise folgendes HTML-Element haben: `<h1 id="Seitenkopffueberschrift" align="center">Text</h1>`, dann können Sie mit `do-`

`document.getElementById()` auf das Element zugreifen und die Objekteigenschaft `align` abfragen oder ändern – z.B.: `alert(document.getElementById('Seitenkopfueberschrift').align)`. Diese JavaScript-Anweisung gibt ein Meldungsfenster aus, in dem center steht, weil `document.getElementById("Seitenkopfueberschrift").align` auf die Eigenschaft `align` des Elementobjekts mit der Id `Seitenkopfueberschrift` zugreift.

In diesem Abschnitt werden für alle HTML-Elemente die vom DOM erlaubten Eigenschaften und Methoden beschrieben. Die Eigenschaften ergeben sich dabei zwangsläufig aus den erlaubten Attributen der HTML-Elemente gemäß HTML 4.0. Die Methoden werden dagegen nur vom DOM festgelegt.

**Beachten Sie:** Jedes HTML-Element stellt gemäß dem DOM außerdem einen **Knoten** im Elementbaum dar. Deshalb gelten für jedes HTML-Element auch alle Eigenschaften und Methoden des `node`-Objekts. Beachten Sie unbedingt die Groß-/Kleinschreibung der Eigenschaften und Methoden, die in diesem Abschnitt zu den einzelnen HTML-Elementobjekten aufgelistet sind. Fehler bei der Groß-/Kleinschreibung führen zu Fehlern in JavaScript. Da Opera 5.12 das Document Object Model nur sehr unvollständig implementiert hat, wird in diesem Dokument jeweils darauf hingewiesen, welche Beispiele auch von diesem Browser interpretiert werden.



### Universaleigenschaften

Universaleigenschaften entsprechen den Universalattributen von HTML. Diese Eigenschaften gelten für fast jedes HTML-Element.

| Eigenschaft            | Status                                          | Bedeutung                           |
|------------------------|-------------------------------------------------|-------------------------------------|
| <code>className</code> | <a href="#">Lesen</a><br><a href="#">Ändern</a> | CSS-Klassenname                     |
| <code>dir</code>       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Schreibrichtung                     |
| <code>id</code>        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | dokumentweit eindeutiger Name       |
| <code>lang</code>      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Landessprache (de, en, fr, it usw.) |
| <code>title</code>     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Titel                               |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p id="p_italiano" lang="it"
  onClick=" alert(document.getElementById('p_italiano').lang)">io senza te</p>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit einer `id="p_italiano"`. Gleichzeitig enthält der Textabsatz einen Event-Handler `onClick`. Beim Anklicken des Textes gibt ein Meldungsfenster aus, um welche Sprache es sich bei dem Text handelt – ausgegeben wird die Wertzuweisung des Attributs `lang`, also der Wert `it`.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.

**a (HTML-Elementobjekt)**

HTML-Elemente `<a>...</a>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft | Status                                          | Bedeutung                              |
|-------------|-------------------------------------------------|----------------------------------------|
| accessKey   | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Hotkey für den Hyperlink               |
| charset     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Zeichensatz des Verweisziels           |
| coords      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | verweissensitive Bereiche bei Objekten |
| href        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweisziel                            |
| hreflang    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Landessprache des Verweisziels         |
| name        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ankername                              |
| rel         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweisziel als »Vorwärtsverknüpfung«  |
| rev         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweisziel als »Rückwärtsverknüpfung« |
| shape       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | verweissensitive Bereiche bei Objekten |
| tabIndex    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Tabulatorreihenfolge für Hyperlinks    |
| target      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Fensteramen des Verweisziels           |
| type        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type des Verweisziels             |

| Methode | Bedeutung                             |
|---------|---------------------------------------|
| blur()  | entfernt den Fokus von diesem Element |
| focus() | setzt den Fokus auf dieses Feld       |

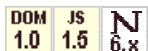
**Ein Beispiel:**

```

<html><head><title>Test</title>
<link rel="stylesheet" href="styles_1.css">
<script type="text/javascript">
<!--
function AndererLink() {
  document.getElementById("selfhtml_link").href = "http://selfaktuell.teamone.de/";
  document.getElementById("selfhtml_link").firstChild.nodeValue = "SELFHTML
aktuell";
}
//-->
</script>
</head><body>
<a id="selfhtml_link" href="http://selfhtml.teamone.de/">SELFHTML</a><br>
<a href="javascript:AndererLink()">anderer Link!</a>
</body></html>

```

Die Beispieldatei enthält zwei Verweise. Der erste Verweis führt auf eine Adresse im Netz, der zweite ruft die JavaScript-Funktion `AndererLink()` auf, die im Dateikopf notiert ist. Diese Funktion weist der Elementeigenschaft `href` einen neuen Wert zu, nämlich eine andere Internetadresse. Mit `getElementById("selfhtml_link")` wird dabei auf den Verweis zugegriffen, der mit `id="selfhtml_link"` ausgezeichnet ist. Außerdem ändert die Funktion `AndererLink()` auch noch dynamisch den Verweistext. Auch dafür wird die DOM-Syntax verwendet, nämlich durch Zuweisung eines neuen Textes an `node.nodeValue` (`firstChild` bezeichnet den ersten Kindknoten eines Knotens).



### **abbr (HTML-Elementobjekt)**

HTML-Elemente `<abbr>...</abbr>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.normal { font-weight:normal }
.fett { font-weight:bold }
-->
</style>
<script type="text/javascript">
<!--
function wechseln() {
  if(document.getElementById("abk").className == "normal")
    document.getElementById("abk").className = "fett";
  else
    document.getElementById("abk").className = "normal";
}
//-->
</script>
</head><body>
<p>Das ist eine <abbr id="abk" class="normal" onMouseOver="wechseln()"
onMouseOut="wechseln()">Abk.</abbr></p>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit der Abkürzung `Abk.`, die mit `<abbr>...</abbr>` ausgezeichnet ist. Das einleitende Tag enthält eine CSS-Klassenangabe, eine eindeutige Id-Angabe und die Event-Handler `onMouseOver` und `onMouseOut`, die jeweils die JavaScript-Funktion `wechseln()` aufrufen, die im Dateikopf notiert ist. Diese Funktion fragt ab, ob der zugewiesene Klassenname `normal` lautet. Wenn ja, wird er auf `fett` geändert, wenn nein, wird er auf `normal` geändert. Dadurch ergibt sich der Effekt, dass beim Überfahren der Abkürzung mit der Maus der Text der Abkürzung `fett` dargestellt wird, ansonsten `normal`.

**Beachten Sie:** Beim Internet Explorer war das Beispiel mit dem `abbr`-Element außer unter dem Internet Explorer 5.0 der Macintosh Edition nicht nachvollziehbar, mit anderen Elementen dagegen schon.

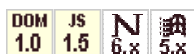
**acronym (HTML-Elementobjekt)**

Die HTML-Elemente `<acronym>...</acronym>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<p>Das ist ein <acronym id="acr"
onMouseOver="document.getElementById('acr').title='Acronym'">Acr.</acronym></p>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit dem Akronym Acr., das mit `<acronym>...</acronym>` ausgezeichnet ist. Das einleitende Tag enthält eine eindeutige Id-Angabe und den Event-Handler `onMouseOver`. Beim Überfahren des Elementtextes mit der Maus wird dadurch dynamisch ein `title`-Attribut gesetzt, das aussagt, was die Abkürzung Acr. bedeutet.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.

**address (HTML-Elementobjekt)**

HTML-Elemente `<address>...</address>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Namen() {
  document.getElementsByTagName("address")[0].firstChild.nodeValue =
  'Stefan Münz';
}
function Text() {
  document.getElementsByTagName("address")[0].firstChild.nodeValue =
  'SELFHTML-Redaktion';
}
//-->
</script>
</head><body>
<div>Dies alles ist von der
<address onMouseOver="Namen()" onMouseOut="Text()">SELFHTML-
Redaktion</address></div>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit der Adressbezeichnung SELFHTML-Redaktion, die mit `<address>...</address>` ausgezeichnet ist. Das einleitende Tag enthält die Event-Handler `onMouseOver` und `onMouseOut`, die jeweils eine JavaScript-Funktion aufrufen. Die Funktion `Namen()`, die bei `onMouseOver` aufgerufen wird, ersetzt den Inhalt SELFHTML-Redaktion durch den Wert Stefan Münz. Die Funktion `Text()` stellt dagegen den Originalzustand wieder her und wird bei `onMouseOut` aufgerufen. Beide Funktionen greifen mit `getElementsByTagName("address")[0]` auf das erste address-Element im Dokument zu. Der jeweils neue Textinhalt des Elements wird mit `node.nodeValue` gesetzt (`firstChild` bezeichnet den ersten Kindknoten eines Knotens).



**applet (HTML-Elementobjekt)**

HTML-Elemente `<applet>...</applet>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                          | Bedeutung                                                  |
|-------------|-------------------------------------------------|------------------------------------------------------------|
| align       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausrichtung                                                |
| alt         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Alternativtext                                             |
| archive     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | kommaseparierte Liste mit Archivdateien                    |
| code        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Klassendatei des Applets                                   |
| codeBase    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Basis-URI des Applets                                      |
| height      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzeigehöhe                                                |
| hspace      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | horizontaler Abstand zwischen Applet und umfließendem Text |
| name        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Name für das Applet                                        |
| object      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Objekt-Id des Applets                                      |
| vspace      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | vertikaler Abstand zwischen Applet und umfließendem Text   |
| width       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzeigebreite                                              |

**Ein Beispiel:**

```

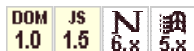
<html><head><title>Test</title>
</head><body>
<applet id="Ticker" code="zticker.class" width="600" height="60">
  <param name="msg" value="Die Energie des Verstehens">
  <param name="speed" value="5">
  <param name="bgco" value="255,255,255">
  <param name="txtco" value="000,000,255">
  <param name="hrefco" value="255,255,255">
</applet>
<form name="Form" action="">
<input type="button" value="40"
onClick="document.getElementById('Ticker').height=40">
<input type="button" value="60"
onClick="document.getElementById('Ticker').height=60">
</form>
</body></html>

```

Das Beispiel enthält ein eingebundenes Java-Applet für einen Lauftext (Ticker). Unterhalb des Applets ist ein Formular mit einem Eingabefeld notiert. Klickt der Anwender auf die Buttons, wird dynamisch die Anzeigehöhe des Java-Applets geändert, und der Lauftext ändert dabei

automatisch seine Größe (sofern das Applet so programmiert ist, dass die Größe des Lauftextes von dem Attribut `height=` im einleitenden `<applet>`-Tag abhängig ist). Mit `document.getElementById('Ticker')` wird auf das Applet zugegriffen, da es im einleitenden Tag das Attribut `id="Ticker"` hat. Geändert wird sein Attribut `height`.

**Beachten Sie:** Der Browser Netscape 6.1 interpretiert dieses Beispiel nicht.



### *area (HTML-Elementobjekt)*

HTML-Elemente `<area>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universal-eigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft            | Status                        | Bedeutung                                  |
|------------------------|-------------------------------|--------------------------------------------|
| <code>accesskey</code> | <b>Lesen</b><br><b>Ändern</b> | Hotkey für den Hyperlink                   |
| <code>alt</code>       | <b>Lesen</b><br><b>Ändern</b> | Kurzbeschreibung des Verweisziels          |
| <code>coords</code>    | <b>Lesen</b><br><b>Ändern</b> | Koordinaten des verweissensitiven Bereichs |
| <code>href</code>      | <b>Lesen</b><br><b>Ändern</b> | Verweisziel                                |
| <code>nohref</code>    | <b>Lesen</b><br><b>Ändern</b> | aktionsloser Bereich                       |
| <code>shape</code>     | <b>Lesen</b><br><b>Ändern</b> | Typ eines verweissensitiven Bereichs       |
| <code>tabindex</code>  | <b>Lesen</b><br><b>Ändern</b> | Tabulatorreihenfolge für Hyperlinks        |
| <code>target</code>    | <b>Lesen</b><br><b>Ändern</b> | Fensteramen des Verweisziels               |

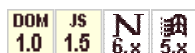
### Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function plus100() {
  document.getElementById("Verweis").coords = "101,101,349,149";
}
//-->
</script>
</head><body>
<map name="Testbild">
<area id="Verweis" shape="rect" coords="1,1,249,49"
      href="javascript:plus100()" title="Koordinaten" alt="Koordinaten">
</map>

</body></html>
```

Das Beispiel enthält eine Grafik, die auf einen `map`-Bereich verweist, in dem ein verweissensitiver Bereich für die Grafik notiert wird. Beim Anklicken dieses verweissensitiven Bereichs wird die JavaScript-Funktion `plus100()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ändert dynamisch die Eigenschaft `coords`, indem sie ihr neue Werte zuweist. Der Effekt ist, dass der verweissensitive Bereich anschließend um 100 Pixel weiter nach rechts unten verlagert wird.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



### **b (HTML-Elementobjekt)**

HTML-Elemente `<b>...</b>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universalieigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<link id="CSS" rel="stylesheet" href="styles_1.css">
</head><body>
<b id="wichtig">wichtige Aussage!</b><br>
<a href="javascript:document.getElementById('wichtig').id='unwichtig';
  alert(document.getElementById('unwichtig').id);">unwichtig machen!</a>
</body></html>
```

Das Beispiel enthält eine mit `<b>...</b>` formatierte wichtige Aussage, bei der im einleitenden Tag das Attribut `id="wichtig"` notiert ist. Unterhalb davon ist ein Verweis notiert, bei dessen Anklicken das `id`-Attribut des `b`-Elements auf den Wert `unwichtig` gesetzt wird. Dazu wird mit `document.getElementById('wichtig').id` auf das Attribut zugegriffen. Anschließend wird mit `document.getElementById('unwichtig').id` auf den neuen `id`-Wert zugegriffen, um den neuen `id`-Wert zur Kontrolle in einem Meldungsfenster auszugeben.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



### **base (HTML-Elementobjekt)**

HTML-Elemente `<base>` haben als DOM-Objekte für den Scriptsprachen-Zugriff die folgenden eigenen Eigenschaften.

| Eigenschaft         | Status                        | Bedeutung                        |
|---------------------|-------------------------------|----------------------------------|
| <code>href</code>   | <b>Lesen</b><br><b>Ändern</b> | Basis-URI                        |
| <code>target</code> | <b>Lesen</b><br><b>Ändern</b> | Default Zielfenster für Verweise |

Ein Beispiel:

```
<html><head><title>Test</title>
<base target="_self">
<script type="text/javascript">
<!--
function blankBase() {
  document.getElementsByTagName("base")[0].target="_blank";
}
```

```
//-->
</script>
</head><body>
<a href="news.htm">News</a><br>
<a href="javascript:blankBase()">Links in neues Fenster laden</a>
</body></html>
```

In der Beispieldatei sind zwei Verweise notiert. Der erste ruft einfach eine andere Datei auf. Normalerweise wird das Verweisziel dabei ins gleiche Fenster geladen. Im Dateikopf ist dies mit `<base id="Zielfenster" target="_self">` auch explizit bestätigt. Der zweite Verweis ruft bei Anklicken jedoch die JavaScript-Funktion `blankBase()` auf, die beim `base`-Element die Eigenschaft `target` auf den Wert `_blank` ändert, was bewirkt, dass Verweisziele dieser Datei in ein neues Fenster geladen werden.

**Beachten Sie:** Mit Netscape (6.1) und unter dem MS Internet Explorer 5.0 Macintosh Edition war das Beispiel mit dem `base`-Element nicht nachvollziehbar. Opera 5.12 interpretiert das Beispiel dagegen. Die Verwendung von Universeigenschaften ist im HTML 4.0-Standard für das `base`-Element nicht vorgesehen. Deshalb sollten Sie bei diesem Element die Methode `document.getElementById()` vermeiden.



### **basefont (HTML-Elementobjekt)**

HTML-Elemente `<basefont>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universeigenschaften sowie die folgenden eigenen Eigenschaften.

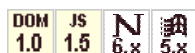
| Eigenschaft | Status                        | Bedeutung          |
|-------------|-------------------------------|--------------------|
| color       | <b>Lesen</b><br><b>Ändern</b> | Basis-Schriftfarbe |
| face        | <b>Lesen</b><br><b>Ändern</b> | Basis-Schriftart   |
| size        | <b>Lesen</b><br><b>Ändern</b> | Basis-Schriftgröße |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function machGross() {
  document.getElementById("ab_hier_anders").size="7";
}
//-->
</script>
</head><body>
<p>Text<br>Text<br>Text<br>Text<br>Text<br>Text<br>Text<br>Text</p>
<basefont id="ab_hier_anders" color="red" size="">
<p>Text<br>Text<br>Text<br>Text<br>Text<br>Text<br>Text<br>Text</p>
<a href="javascript:machGross()">nicht nur rot, sondern auch gross</a>
</body></html>
```

Das Beispiel enthält Text und zwischendrin ein `basefont`-Element. Am Ende steht ein Verweis, bei dessen Anklicken die JavaScript-Funktion `machGross()` aufgerufen wird. Mit `getElementById("ab_hier_anders")` greift diese Funktion auf das `basefont`-Element zu und ändert dessen Eigenschaft `size` auf den Wert 7.

**Beachten Sie:** Mit Netscape (6.1) war das Beispiel mit dem `basefont`-Element nicht nachvollziehbar.



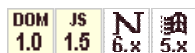
### ***bdo (HTML-Elementobjekt)***

HTML-Elemente `<bdo>...</bdo>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function abc2cba() {
  document.getElementsByTagName("bdo")[0].dir="rtl";
}
//-->
</script>
</head><body>
<bdo dir="ltr">ABCDEFGHIJKLMNOPQRSTUVWXYZ</bdo><br>
<a href="javascript:abc2cba()">umkehren!</a>
</body></html>
```

Das Beispiel enthält ein `bdo`-Element mit dem Attribut `dir="ltr"` (Schriftrichtung von links nach rechts), das das Alphabet in Großbuchstaben als Inhalt hat. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die JavaScript-Funktion `abc2cba()` aufgerufen. Diese greift mit `getElementsByTagName("bdo")[0]` auf das erste `bdo`-Element im Dokument zu und ändert die Schriftrichtung auf `rtl`, also von rechts nach links. Das Großbuchstabenalphabet wird dadurch umgedreht.

**Beachten Sie:** Unter dem MS Internet Explorer 5.0 Macintosh Edition war das Beispiel nicht nachvollziehbar.



### ***big (HTML-Elementobjekt)***

HTML-Elemente `<big>...</big>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.verybig { font-size:300% }
-->
</style>
<script type="text/javascript">
<!--
function nochmal() {
  document.getElementById("biggie").className = "verybig";
}
```

```

}
//-->
</script>
</head><body>
<p><big id="biggie" onClick="nochmehr()">gross und stark!</big></p>
</body></html>

```

Das Beispiel enthält in einem Textabsatz Text, der mit `<big>...</big>` ausgezeichnet ist. Im einleitenden `<big>`-Tag ist der Event-Handler `onClick` notiert. Beim Anklicken des Textes dieses Elements wird die JavaScript-Funktion `nochmehr()` aufgerufen. Diese greift mit `document.getElementById("biggie")` auf das `big`-Element zu und weist ihm die im Dateikopf in einem Style-Bereich definierte Klasse `verybig` zu. Der Text wird dadurch auf 300% seiner normalen Größe vergrößert.



### **blockquote (HTML-Elementobjekt)**

HTML-Elemente `<blockquote>...</blockquote>` haben als DOM-Objekte für den Script-sprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft       | Status                                                                            | Bedeutung           |
|-------------------|-----------------------------------------------------------------------------------|---------------------|
| <code>cite</code> |  | URI der Zitatquelle |

Ein Beispiel:

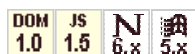
```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function showCite() {
  alert(document.getElementById('w3zitat').cite);
}
//-->
</script>
</head><body>
<p>Das W3-Konsortium schreibt über das DOM:</p>
<blockquote id="w3zitat"
  cite="http://www.w3.org/TR/REC-DOM-Level-1/introduction.html"
  onMouseOver="showCite()">
The Document Object Model (DOM) is an application programming
interface (API) for valid HTML and well-formed XML documents.
</blockquote>
</body></html>

```

Das Beispiel enthält ein mit `<blockquote>...</blockquote>` ausgezeichnetes Zitat, in dessen einleitendem Tag auch der URI der Quelle mit dem Attribut `cite=` angegeben ist. Ferner ist dort der Event-Handler `onMouseOver` notiert, der bewirkt, dass beim Überfahren des Zitats mit der Maus die JavaScript-Funktion `showCite()` aufgerufen wird. Diese greift mit `document.getElementById('w3zitat')` auf das `blockquote`-Element zu und gibt in einem Meldungsfenster den Wert des `cite`-Attributs aus.

**Beachten Sie:** Opera 5.12 gibt in diesem Beispiel `undefined` zurück.



### **body (HTML-Elementobjekt)**

HTML-Elemente `<body>...</body>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft             | Status                        | Bedeutung                                                    |
|-------------------------|-------------------------------|--------------------------------------------------------------|
| <code>aLink</code>      | <b>Lesen</b><br><b>Ändern</b> | dokumentweite Farbe für aktivierte Links                     |
| <code>background</code> | <b>Lesen</b><br><b>Ändern</b> | URI einer dokumentweiten Hintergrundgrafik                   |
| <code>bgColor</code>    | <b>Lesen</b><br><b>Ändern</b> | dokumentweite Hintergrundfarbe                               |
| <code>link</code>       | <b>Lesen</b><br><b>Ändern</b> | dokumentweite Farbe für Links zu noch nicht besuchten Seiten |
| <code>text</code>       | <b>Lesen</b><br><b>Ändern</b> | dokumentweite Textfarbe                                      |
| <code>vLink</code>      | <b>Lesen</b><br><b>Ändern</b> | dokumentweite Farbe für Links zu bereits besuchten Seiten    |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function WerteSetzen() {
  document.getElementsByTagName("body")[0].text = document.Formular.Text.value;
  document.getElementsByTagName("body")[0].link = document.Formular.Link.value;
  document.getElementsByTagName("body")[0].vLink = document.Formular.VLink.value;
  document.getElementsByTagName("body")[0].bgColor =
document.Formular.BgColor.value;
}
//-->
</script>
</head><body>
<h1>Ein dynamisches Dokument</h1>
<a href="news.htm"><b>Ein Link zu den News</b></a>

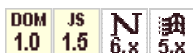
<form name="Formular" action="">
<pre>
Textfarbe:      <input type="text" size="7" name="Text">
Linkfarbe:      <input type="text" size="7" name="Link">
Linkfarbe (besucht): <input type="text" size="7" name="VLink">
Hintergrundfarbe: <input type="text" size="7" name="BgColor">
Einstellungen:  <input type="button" value="Testen!" onClick="WerteSetzen()">
</pre>
</form>

</body></html>
```

Die Beispieldatei enthält im `body`-Bereich eine Überschrift, einen Link und ein Formular mit verschiedenen Eingabefeldern. In den Eingabefeldern kann der Anwender neue Basisfarben für das Dokument einstellen – typische Eingabewerte sind also hexadezimale Angaben wie `#FFFFCC` oder Farbwörter wie `maroon`. Beim Anklicken des Buttons mit der Aufschrift `Testen` wird die JavaScript-Funktion `WerteSetzen()` aufgerufen, die im Dateikopf notiert ist. Diese greift mit `document.getElementsByTagName("body")[0]` auf das »erste `body`-Element« der Datei zu und weist den entsprechenden Eigenschaften die eingegebenen Werte aus dem Formular zu. Dadurch verändern sich die Basisfarben des Dokuments.

**Beachten Sie:** Sie können im MS Internet Explorer und im Netscape 6 zusätzlich auf die Eigenschaften `offsetTop`, `offsetLeft`, `offsetWidth`, `offsetHeight`, `offsetParent` und `innerHTML` zugreifen. Im Netscape 6 stehen diese Eigenschaften jedoch erst nach dem Laden des Dokumentes zur Verfügung.

Beim MS Internet Explorer sind auf das `body`-Objekt die meisten Eigenschaften des `all`-Objekts anwendbar. Der Grund ist, dass `body` im Internet Explorer schon vor Einführung der DOM-Syntax als Objektname existierte. Unter dem MS Internet Explorer 5.0 Macintosh Edition war das Beispiel nicht vollständig nachvollziehbar und führte teilweise zu einem sehr seltsamen Verhalten.



### **br (HTML-Elementobjekt)**

HTML-Elemente `<br>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft        | Status                        | Bedeutung                            |
|--------------------|-------------------------------|--------------------------------------|
| <code>clear</code> | <b>Lesen</b><br><b>Ändern</b> | Fortsetzungsposition bei Textumfluss |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>

Dieser Text fließt um das Logo herum,
weil dies im align-Attribut des Logos so angegeben ist.<br id="Umbruch">
Gilt das auch bei diesem Text?
</body></html>
```

Das Beispiel enthält eine Grafik, bei der durch die Angabe `align="left"` festgelegt wird, dass der nachfolgende Text rechts um die Grafik fließt. Der Text enthält einen Zeilenumbruch, markiert durch `<br>`. Die Grafik enthält einen Event-Handler `onClick`, der bewirkt, dass beim Anklicken der Grafik mit `document.getElementById('Umbruch')` auf das Zeilenumbruch-Element zugegriffen wird. Ihm wird die Eigenschaft `clear` mit dem Wert `all` zugewiesen. Der Text unterhalb des Umbruchs rutscht bei einem Klick auf die Grafik unter die Grafik, da `<br clear="all">` die Textfortsetzung unterhalb der Grafik bewirkt.



**button (HTML-Elementobjekt)**

HTML-Elemente `<button>...</button>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                         |
|-------------|-------------------------------|-----------------------------------|
| accessKey   | <b>Lesen</b><br><b>Ändern</b> | Hotkey für den Tastaturzugriff    |
| disabled    | <b>Lesen</b><br><b>Ändern</b> | Button kann nicht betätigt werden |
| form        | <b>Lesen</b>                  | zugehöriges Formularelement       |
| name        | <b>Lesen</b><br><b>Ändern</b> | Name für den Button               |
| tabIndex    | <b>Lesen</b><br><b>Ändern</b> | Tabulatorreihenfolge              |
| type        | <b>Lesen</b>                  | Typ des Buttons                   |
| value       | <b>Lesen</b><br><b>Ändern</b> | Absendewert des Buttons           |

Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
function ZeitAufButton() {
  var jetzt = new Date();
  var Zeit = jetzt.getTime();
  document.getElementsByName("Zeitbutton")[0].value = Zeit;
  document.getElementsByName("Zeitbutton")[0].firstChild.nodeValue = Zeit;
}
//-->
</script>
</head><body>
<form name="Formular" action="">
<button name="Zeitbutton" value="" onClick="ZeitAufButton()">Zeit!</button>
</form>
</body></html>
```

Das Beispiel definiert in einem Formular einen Button, der den Event-Handler `onClick` enthält. Beim Anklicken des Buttons wird deshalb die JavaScript-Funktion `ZeitAufButton()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ermittelt mit Hilfe des `Date`-Objekts den aktuellen Zeitpunkt in Millisekunden und weist diesen ermittelten Wert sowohl dem Absendewert des Buttons als auch dessen Aufschrift zu. Dazu wird mit `document.getElementsByName("Zeitbutton")[0]` auf das erste Element mit dem Attribut `name="Zeitbutton"` zugegriffen. Der Absendewert wird durch Zuweisen des Wertes von `Zeit` an die Eigenschaft `value` geändert. Für die Button-Aufschrift muss der Wert des ersten Kindknotens (`firstChild.nodeValue`) des Buttons geändert werden. Das Beispiel bewirkt, dass bei jedem Anklicken des Buttons die Millisekundenzeit auf der Button-Aufschrift aktualisiert wird.

**caption (HTML-Elementobjekt)**

HTML-Elemente `<caption>...</caption>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft | Status                        | Bedeutung   |
|-------------|-------------------------------|-------------|
| align       | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Wechseln() {
  if(document.getElementById("THeader").align == "left")
    document.getElementById("THeader").align = "right";
  else
    document.getElementById("THeader").align = "left";
}
//-->
</script>
</head><body>
<table border="5" cellspacing="4">
<caption id="THeader" align="left"
onClick="Wechseln()"><b>Spielstand:</b></caption>
<tr>
<td>Hans Moosreiner:</td><td>5 Punkte</td>
</tr><tr>
<td>Lisa Wohlthu:</td><td>3 Punkte</td>
</tr>
</table>
</body></html>
```

Das Beispiel enthält eine kleine Tabelle mit einer Tabellenüberschrift, die durch `<caption>...</caption>` ausgezeichnet ist. Das einleitende Tag enthält ferner den Event-Handler `onClick`. Dadurch wird beim Anklicken der Tabellenüberschrift die JavaScript-Funktion `Wechseln()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("THeader")` auf die Tabellenüberschrift zu und fragt ab, ob deren `align`-Eigenschaft den Wert `left` hat. Wenn ja, wird er auf `right` geändert, wenn nein (also wenn er auf `right` gesetzt ist), wird er wieder auf `left` gesetzt. Auf diese Weise springt die Tabellenüberschrift bei jedem Anklicken zur anderen Seite (links bzw. rechts).

**Beachten Sie:** Mit Netscape (6.1) und dem MS Internet Explorer 5.0 Macintosh Edition war das Beispiel mit dem `caption`-Element nicht nachvollziehbar.

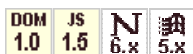
**center (HTML-Elementobjekt)**

HTML-Elemente `<center>...</center>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<center id="zentriert" title="Das ist wirklich zentriert!"
onMouseover="alert(document.getElementById('zentriert').title)">
<h1>Das ist zentriert</h1>
<h2>Das ist zentriert</h2>
<h3>Das ist zentriert</h3>
</center>
</body></html>
```

Das Beispiel enthält einen mit `<center>...</center>` ausgezeichneten Bereich, innerhalb dessen alles zentriert dargestellt wird. Im einleitenden Tag ist der Event-Handler `onMouseOver` notiert. Beim Überfahren des zentrierten Bereichs wird ein Meldungsfenster ausgegeben, das den Wert des Attributs `title` ausgibt. Dazu wird mit `document.getElementById('zentriert')` auf das Element zugegriffen.

**Beachten Sie:** Opera 5.12 interpretiert dieses Beispiel ebenfalls.

**cite (HTML-Elementobjekt)**

HTML-Elemente `<cite>...</cite>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<cite id="Zitat" title="Zitat von Kafka"
onMouseOver="alert(document.getElementById('Zitat').title)">
Die Krähen behaupten, eine einzige Krähe könne den Himmel zerstören.
Das ist zweifellos, beweist aber nichts gegen den Himmel, denn Himmel bedeutet
eben:
Unmöglichkeit von Krähen.
</cite>
</body></html>
```

Das Beispiel enthält einen mit `<cite>...</cite>` ausgezeichneten Bereich. Im einleitenden Tag ist der Event-Handler `onMouseOver` notiert. Beim Überfahren des zentrierten Bereichs wird ein Meldungsfenster ausgegeben, das den Wert des Attributs `title` ausgibt. Dazu wird mit `document.getElementById('Zitat')` auf das Element zugegriffen.

**Beachten Sie:** Opera 5.12 interpretiert dieses Beispiel ebenfalls.

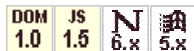
**code (HTML-Elementobjekt)**

HTML-Elemente `<code>...</code>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften.

Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.farbig { color:blue }
-->
</style>
<script language="JavaScript" type="text/javascript">
<!--
function farbig() {
  document.getElementById("Perlbeispiel").className = "farbig";
}
//-->
</script>
</head><body>
<p><code id="Perlbeispiel" onClick="farbig()">$Text =~ s/[a-z]| //gi;</code></p>
</body></html>
```

Das Beispiel enthält in einem Textabsatz Perl-Code, der mit `<code>...</code>` ausgezeichnet ist. Im einleitenden `<code>`-Tag ist der Event-Handler `onClick` notiert. Beim Anklicken des Textes dieses Elements wird die JavaScript-Funktion `farbig()` aufgerufen. Diese greift mit `document.getElementById("Perlbeispiel")` auf das `code`-Element zu und weist ihm die im Dateikopf in einem Style-Bereich definierte Klasse `farbig` zu. Der Text wird dadurch blau dargestellt.



### **col (HTML-Elementobjekt)**

HTML-Elemente `<col>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universal-eigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                              |
|-------------|-------------------------------|----------------------------------------|
| align       | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung                            |
| ch          | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen                    |
| chOff       | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens      |
| span        | <b>Lesen</b><br><b>Ändern</b> | Anzahl Spalten, für die Angaben gelten |
| vAlign      | <b>Lesen</b><br><b>Ändern</b> | Tabellenspalten vertikal ausrichten    |
| width       | <b>Lesen</b><br><b>Ändern</b> | Spaltenbreite                          |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function gleichbreit() {
```

```

for(i = 0; i < document.getElementsByTagName("col").length; i++)
    document.getElementsByTagName("col")[i].width = "320";
}
//-->
</script>
</head><body>
<table border="1">
    <colgroup>
        <col width="80">
        <col width="100">
        <col width="320">
    </colgroup>
    <tr>
        <td>1. Zeile, 1. Spalte</td>
        <td>1. Zeile, 2. Spalte</td>
        <td>1. Zeile, 3. Spalte</td>
    </tr>
</table>
<a href="javascript:gleichbreit()">Spalten gleich breit machen!</a>
</body></html>

```

Das Beispiel enthält eine Tabelle mit drei Spalten, deren Breiten mit `col`-Elementen unterschiedlich vordefiniert werden. Unterhalb der Tabelle ist ein Verweis notiert, bei dessen Anklicken die JavaScript-Funktion `gleichbreit()` aufgerufen wird. Diese greift in einer `for`-Schleife der Reihe nach mit `document.getElementsByTagName("col")` auf die einzelnen `col`-Elemente zu und weist ihnen jeweils den Wert 320 für die Eigenschaft `width` zu. Dadurch wird die Tabelle dynamisch neu angezeigt, und zwar mit drei gleich breiten Spalten.

**Beachten Sie:** Unter dem MS Internet Explorer 5.0 Macintosh Edition war das Beispiel nicht nachvollziehbar.



### **colgroup (HTML-Elementobjekt)**

HTML-Elemente `<colgroup>...</colgroup>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

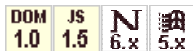
| Eigenschaft         | Status                        | Bedeutung                              |
|---------------------|-------------------------------|----------------------------------------|
| <code>align</code>  | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung                            |
| <code>ch</code>     | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen                    |
| <code>chOff</code>  | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens      |
| <code>span</code>   | <b>Lesen</b><br><b>Ändern</b> | Anzahl Spalten, für die Angaben gelten |
| <code>vAlign</code> | <b>Lesen</b><br><b>Ändern</b> | Tabellenspalten vertikal ausrichten    |
| <code>width</code>  | <b>Lesen</b><br><b>Ändern</b> | Spaltenbreite                          |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function ausrichten() {
    document.getElementById("Tabellenspalten").align = "right";
}
//-->
</script>
</head><body>
<table border="1">
  <colgroup id="Tabellenspalten">
    <col width="200">
  </colgroup>
  <tr><td>Hansi</td></tr>
  <tr><td>Willi</td></tr>
  <tr><td>Manni</td></tr>
</table>
<a href="javascript:ausrichten()">ausrichten!</a>
</body></html>
```

Das Beispiel enthält eine Tabelle mit einer `colgroup`-Definition. Unterhalb der Tabelle ist ein Verweis notiert, bei dessen Anklicken die JavaScript-Funktion `ausrichten()` aufgerufen wird. Diese greift mit `document.getElementById("Tabellenspalten")` auf das `colgroup`-Element zu und weist ihm den Wert `right` für die Eigenschaft `align` zu. Dadurch wird die Tabelle dynamisch neu angezeigt, und alle Zellen werden rechtsbündig ausgerichtet.

**Beachten Sie:** Mit Netscape (6.1) und dem MS Internet Explorer 5.0 Macintosh Edition war das Beispiel nicht nachvollziehbar. Aber auch der Internet Explorer 5.x interpretiert die Eigenschaften von `colgroup` nur unvollständig.



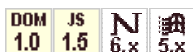
### **dd (HTML-Elementobjekt)**

HTML-Elemente `<dd>...</dd>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Titel() {
    for(var i = 0; i < document.getElementsByTagName("dt").length; i++)
        document.getElementsByTagName("dd")[i].title =
            document.getElementsByTagName("dt")[i].firstChild.nodeValue;
}
//-->
</script>
</head><body>
<dl>
<dt>*.bmp</dt><dd>Bitmap-Grafiken</dd>
<dt>*.html</dt><dd>Web-Seiten-Dateien</dd>
```

```
<dt>*.css</dt><dd>Style-Dateien für Web-Seiten</dd>
</dl>
<a href="javascript:Titel()">Titel</a>
</body></html>
```

Das Beispiel enthält eine Definitionsliste mit drei Einträgen. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die JavaScript-Funktion `Titel()` aufgerufen, die im Dateikopf notiert ist. Diese greift in einer `for`-Schleife der Reihe nach auf alle `dt`-Elemente zu und weist den zugehörigen `dd`-Elementen den Inhalt der `dt`-Elemente als `title`-Attribut zu.



### ***del*** (HTML-Elementobjekt)

HTML-Elemente `<del>...</del>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

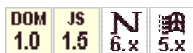
| Eigenschaft           | Status                                                                            | Bedeutung                      |
|-----------------------|-----------------------------------------------------------------------------------|--------------------------------|
| <code>cite</code>     |  | URI für Gründe der Änderung    |
| <code>dateTime</code> |  | Datum und Uhrzeit der Änderung |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Grund() {
  document.getElementById("HTMLVersion").cite = "http://www.w3.org/TR/html401";
}
//-->
</script>
</head><body onLoad="Grund()">
<p><del id="HTMLVersion"
onMouseOver="alert(document.getElementById('HTMLVersion').cite)">
die aktuelle HTML-Version ist 2.0</del></p>
</body></html>
```

Das Beispiel enthält eine mit `<del>...</del>` gelöschte Änderungsmarkierung. Nachdem die Datei geladen ist (Event-Handler `onLoad` im einleitenden `<body>`-Tag), wird die JavaScript-Funktion `Grund()` aufgerufen, die im Dateikopf notiert ist. Diese greift mit `document.getElementById("HTMLVersion")` auf das `del`-Element zu und weist ihm die im HTML-Code nicht notierte Eigenschaft `cite` mit einem Wert zu. Das `del`-Element verfügt zur Kontrolle über einen Event-Handler `onMouseOver`. Beim Überfahren des markierten Textes wird dadurch ein Meldungsfenster angezeigt, das den aktuellen Wert der Eigenschaft `cite` ausgibt.

**Beachten Sie:** Opera 5.12 erlaubt nur einen lesenden Zugriff auf diese Eigenschaft.

**dfn (HTML-Elementobjekt)**

HTML-Elemente `<dfn>...</dfn>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p><dfn id="Definition" title="Dies ist eine Definition!"
onMouseOver="alert(document.getElementById('Definition').title)">
Ein Uhu ist ein Vogel und kein Kleber</dfn></p>
</body></html>
```

Das Beispiel enthält eine mit `<dfn>...</dfn>` ausgezeichnete Definition. Beim Überfahren mit der Maus (`onMouseOver`) wird ein Meldungsfenster ausgegeben, das den Wert des `title`-Attributs des `dfn`-Elements ausgibt. Dazu wird mit `document.getElementById('Definition')` auf das Element zugegriffen.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.

**dir (HTML-Elementobjekt)**

HTML-Elemente `<dir>...</dir>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft | Status                        | Bedeutung                    |
|-------------|-------------------------------|------------------------------|
| compact     | <b>Lesen</b><br><b>Ändern</b> | Darstellung in enger Schrift |

Ein Beispiel:

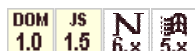
```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function zeigKompakt() {
  document.getElementsByTagName("dir")[0].compact = true;
}
//-->
</script>
</head><body>
<dir onMouseOver="zeigKompakt()">
<li>Eigene Dateien</li>
<li>Programme</li>
<li>Windows</li>
<li>temp</li>
</dir>
</body></html>
```

Das Beispiel enthält eine Verzeichnisliste. Beim Überfahren der Liste mit der Maus (`onMouseOver`) wird die JavaScript-Funktion `zeigKompakt()` aufgerufen, die im Dateikopf notiert ist.



Die Funktion greift mit `document.getElementsByTagName("div")[0]` auf das erste `div`-Element in der Datei zu und setzt seine Eigenschaft `compact` auf den Wert `true` (»wahr«). Dadurch wird die gesamte Liste in kompakter Schrift dargestellt.

**Beachten Sie:** Weder Netscape noch Internet Explorer interpretieren das HTML-Attribut `compact`.



### *div (HTML-Elementobjekt)*

HTML-Elemente `<div>...</div>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft        | Status                                                                            | Bedeutung   |
|--------------------|-----------------------------------------------------------------------------------|-------------|
| <code>align</code> |  | Ausrichtung |

Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
div { border:solid 3px red; padding:10px; }
-->
</style>

<script type="text/javascript">
<!--
function ausrichten() {
  document.getElementById("zweiter").align = "center";
  document.getElementById("dritter").align = "right";
}
//-->
</script>
</head><body>
<div id="erster"><h1>Erster Bereich</h1><p>mit etwas Text</p></div>
<hr>
<div id="zweiter"><h1>Zweiter Bereich</h1><p>mit etwas Text</p></div>
<hr>
<div id="dritter"><h1>Dritter Bereich</h1><p>mit etwas Text</p></div>
<a href="javascript:ausrichten()">ausrichten!</a>
</body></html>
```

Das Beispiel enthält insgesamt drei `div`-Bereiche. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die JavaScript-Funktion `ausrichten()` aufgerufen. Diese Funktion ändert für den zweiten und dritten der Bereiche die Ausrichtung auf »zentriert« bzw. »rechts«. Dazu wird mit `document.getElementById("zweiter")` auf den zweiten Bereich zugegriffen und mit `document.getElementById("dritter")` auf den dritten. Alle Elemente, die innerhalb der Bereiche notiert sind, werden dynamisch neu ausgerichtet.

**DOM 1.0** **dl (HTML-Elementobjekt)**

HTML-Elemente `<dl>...</dl>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft | Status                                                                            | Bedeutung                    |
|-------------|-----------------------------------------------------------------------------------|------------------------------|
| compact     |  | Darstellung in enger Schrift |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function zeigKompakt() {
  document.getElementsByTagName("dl")[0].compact = true;
}
//-->
</script>
</head><body>
<dl onMouseOver="zeigKompakt()">
<dt>User' Guide</dt><dd>Benutzerhandbuch</dd>
<dt>User Instructions</dt><dd>Bedienungsanleitung</dd>
</dl>
</body></html>
```

Das Beispiel enthält eine Definitionsliste. Beim Überfahren der Liste mit der Maus (onMouseOver) wird die JavaScript-Funktion `zeigKompakt()` aufgerufen, die im Dateikopf notiert ist. Die Funktion greift mit `document.getElementsByTagName("dl")[0]` auf das erste dl-Element in der Datei zu und setzt seine Eigenschaft `compact` auf den Wert `true` (»wahr«). Dadurch wird die gesamte Liste in kompakter Schrift dargestellt.

**Beachten Sie:** Weder Netscape noch Internet Explorer interpretieren das HTML-Attribut `compact`.

**DOM 1.0 JS 1.5 5.x** **dt (HTML-Elementobjekt)**

HTML-Elemente `<dt>...</dt>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.hervorgehoben { font-weight:bold; color:red; }
-->
</style>
<script type="text/javascript">
<!--
function Wechseln(x) {
```

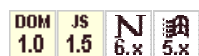
```

if(document.getElementsByTagName("dt")[x].className == "")
    document.getElementsByTagName("dt")[x].className = "hervorgehoben";
else
    document.getElementsByTagName("dt")[x].className = "";
}
//-->
</script>
</head><body>
<dl>
<dt class="" onClick="Wechseln(0)">User' Guide</dt><dd>Benutzerhandbuch</dd>
<dt class="" onClick="Wechseln(1)">User
Instructions</dt><dd>Bedienungsanleitung</dd>
</dl>
</body></html>

```

Das Beispiel enthält eine Definitionsliste. Jedes der dt-Elemente enthält eine leere Style-Klassenzuweisung und den Event-Handler `onClick`. Beim Anklicken der Elemente wird jeweils die JavaScript-Funktion `Wechseln()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementsByTagName("dt")[x]` auf das jeweilige Element zu – `x` ist dabei ein Parameter dafür, das wievielte Element dieses Typs in der Datei gemeint ist. Die dt-Elemente übergeben der Funktion den entsprechenden Parameter beim Aufruf mit `Wechseln(0)` bzw. `Wechseln(1)`. Das Beispiel bewirkt, dass dem betroffenen dt-Element beim ersten Anklicken der Klassenname `hervorgehoben` zugewiesen wird und beim nächsten Anklicken wieder eine leere Klasse. Die Klasse `hervorgehoben`, die im Dateikopf in einem Style-Bereich definiert ist, bewirkt, dass das Element fett und rot dargestellt wird.

**Beachten Sie:** Netscape 6.1 interpretiert dieses Beispiel nicht.



### *em (HTML-Elementobjekt)*

HTML-Elemente `<em>...</em>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universalieigenschaften. Ein Beispiel:

```

<html><head><title>Test</title>
<style type="text/css">
<!--
.mitVielGefuehl { font-family:Lucida Calligraphy; font-size:150%; color:maroon; }
-->
</style>
<script type="text/javascript">
<!--
function xem() {
    document.getElementById("Herzschmerz").className = "mitVielGefuehl";
}
function em() {
    document.getElementById("Herzschmerz").className = "";
}
//-->
</script>
</head><body>

```

```
<p>Das ist Text, der sich
<em id="Herzschmerz" onMouseOver="xem()" onMouseOut="em()">vor Herzschmerz kaum
retten kann</em>.</p>
</body></html>
```

Das Beispiel enthält einen Satz, von dem ein Teil mit `<em>...</em>` ausgezeichnet ist. Das einleitende `<em>`-Tag enthält die Event-Handler `onMouseOver` und `onMouseOut`. In einem Fall wird die JavaScript-Funktion `xem()` aufgerufen, im anderen die Funktion `em()`. Beide Funktionen greifen mit `document.getElementById("Herzschmerz")` auf das `em`-Element zu und weisen einmal den Style-Klassennamen `mitVielGefuehl` und einmal einen leeren Klassennamen zu. Die CSS-Klasse `mitVielGefuehl` ist im Dateikopf definiert. Beim Überfahren des `em`-Elements mit der Maus wird der Text im Element dynamisch stark hervorgehoben.

### **fieldset (HTML-Elementobjekt)**

HTML-Elemente `<fieldset>...</fieldset>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

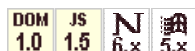
| Eigenschaft | Status       | Bedeutung                   |
|-------------|--------------|-----------------------------|
| form        | <b>Lesen</b> | zugehöriges Formularelement |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Hinweis() {
  if(document.getElementById("Absender").form != null)
    alert("Geben Sie Vornamen und Zunamen in jedem Fall an!");
}
//-->
</script>
</head><body bgcolor="#EEEEEE">
<form name="Formular" action="">
<fieldset id="Absender" onMouseOver="Hinweis()">
<legend>Absender</legend>
<table><tr>
  <td align="right">Vorname:</td>
  <td><input type="text" size="40" maxlength="40"></td>
</tr><tr>
  <td align="right">Zuname:</td>
  <td><input type="text" size="40" maxlength="40"></td>
</tr></table>
</fieldset>
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einem `fieldset`-Bereich. Im einleitenden `<fieldset>`-Tag ist der Event-Handler `onMouseOver` notiert, der die JavaScript-Funktion `Hinweis()` aufruft, die im Dateikopf definiert ist. Mit `if(document.getElementById("Absender").form != null)` fragt die Funktion ab, ob die Eigenschaft `form` einen Wert hat. Wenn der Wert ungleich `null` ist, bedeutet dies, dass das `fieldset`-Element wie vorgeschrieben innerhalb eines Formulars vorkommt. In diesem Fall wird eine Meldung ausgegeben.

**Beachten Sie:** Der MS Internet Explorer interpretiert die Eigenschaft `form` erst ab Version 6.0.



### font (HTML-Elementobjekt)

HTML-Elemente `<font>...</font>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

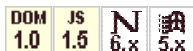
| Eigenschaft | Status                                          | Bedeutung    |
|-------------|-------------------------------------------------|--------------|
| color       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Schriftfarbe |
| face        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Schriftart   |
| size        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Schriftgröße |

Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
var sizes = new Array ("1","2","3","4","5","6","7");
var colors = new Array("#FF0000","#0000FF","#009900","#CC00CC");
var faces = new Array("Arial","Tahoma","Wide Latin");
function Spielen() {
  var s = parseInt((Math.random() * 100) % 7);
  var c = parseInt((Math.random() * 100) % 4);
  var f = parseInt((Math.random() * 100) % 3);
  var i = parseInt((Math.random() * 100) % 7);
  document.getElementsByTagName("font")[i].size = sizes[s];
  document.getElementsByTagName("font")[i].color = colors[c];
  document.getElementsByTagName("font")[i].face = faces[f];
  window.setTimeout("Spielen()",1000);
}
//-->
</script>
</head><body onLoad="Spielen()">
<p>
<font>So viel verspielter Text, o Sophie!</font><br>
<font>So viel verspielter Text, o Sophie!</font><br>
<font>So viel verspielter Text, o Sophie!</font><br>
<font>So viel verspielter Text, o Sophie!</font><br>
<font>So viel verspielter Text, o Sophie!</font><br>
```

```
<font>So viel verspielter Text, o Sophie!</font><br>
<font>So viel verspielter Text, o Sophie!</font><br>
</p>
</body></html>
```

Das Beispiel enthält insgesamt sieben Texte, die mit `font`-Elementen ausgezeichnet sind. Keines der `font`-Elemente enthält allerdings in HTML irgendwelche Attribute. Nachdem das Dokument geladen ist (Event-Handler `onLoad` im einleitenden `<body>`-Tag), wird die JavaScript-Funktion `Spiele()` aufgerufen. Zuvor wurden bereits diverse Arrays angelegt. Die Funktion `Spiele()` ermittelt mittels `Math.random()` Zufallszahlen und trimmt diese mit Hilfe von Multiplikation mit 100, Modulodivision und Anwendung der Funktion `parseInt()` so, dass ganzzahlige Zufallszahlen erzeugt werden. Diese werden als Index für die zuvor definierten Arrays verwendet. Solchermaßen ausgerüstet, greift die Funktion mit `document.getElementsByTagName("font")[i]` zufällig auf eines der `font`-Elemente zu und weist ihm die ebenfalls zufällig ermittelten Werte aus den Arrays für die Eigenschaften `size`, `color` und `face` zu. Am Ende wartet die Funktion eine Sekunde (`window.setTimeout()`) und ruft sich dann selber wieder auf. Dadurch ergibt sich ein endloses Spiel an zufälligen Textänderungen.



### form (HTML-Elementobjekt)

HTML-Elemente `<form>...</form>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft                | Status                                          | Bedeutung                                  |
|----------------------------|-------------------------------------------------|--------------------------------------------|
| <code>acceptCharset</code> | <a href="#">Lesen</a><br><a href="#">Ändern</a> | unterstützte Zeichensätze                  |
| <code>action</code>        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | URI für die Verarbeitung der Formulardaten |
| <code>elements</code>      | <a href="#">Lesen</a>                           | Array für die zugehörigen Formularelemente |
| <code>enctype</code>       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type für Formulardatenübertragung     |
| <code>length</code>        | <a href="#">Lesen</a>                           | Anzahl Formularelemente                    |
| <code>name</code>          | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Name des Formulars                         |
| <code>method</code>        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Übertragungsmethode für Formulardaten      |
| <code>target</code>        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Fenstername für Antwortausgaben            |

| Methode               | Bedeutung             |
|-----------------------|-----------------------|
| <code>reset()</code>  | Formular zurücksetzen |
| <code>submit()</code> | Formular absenden     |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Formular" action="datei.htm" method="get">
<p>
  <input name="Text" type="text"><br>
  [<a
href="javascript:document.getElementsByName('Formular')[0].submit()"><b>Absenden</b>
</a>]
  [<a
href="javascript:document.getElementsByName('Formular')[0].reset()"><b>Abbrechen</b>
</a>]
</p>
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit zwei Verweisen anstelle der sonst üblichen Submit- und Reset-Buttons. Der eine Verweis führt die `submit()`-Methode aus, der andere die `reset()`-Methode. Beide Verweise greifen dazu mit `document.getElementsByName('Formular')[0]` auf das erste Element mit dem Namen `Formular` zu.

**Beachten Sie:** Dieses Beispiel funktioniert auch in Opera 5.12. In lokaler Umgebung außerhalb des HTTP-Protokolls (`file:`) wird jedoch nach dem Absenden der Quellcode der Zielfeile angezeigt.



### frame (HTML-Elementobjekt)

HTML-Elemente `<frame>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universal-eigenschaften sowie die folgenden eigenen Eigenschaften.

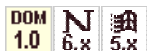
| Eigenschaft               | Status                        | Bedeutung                                                                          |
|---------------------------|-------------------------------|------------------------------------------------------------------------------------|
| <code>frameBorder</code>  | <b>Lesen</b><br><b>Ändern</b> | Rahmen sichtbar oder nicht sichtbar                                                |
| <code>longDesc</code>     | <b>Lesen</b><br><b>Ändern</b> | URI für längere Beschreibung zum Inhalt des Frame-Fensters                         |
| <code>marginHeight</code> | <b>Lesen</b><br><b>Ändern</b> | Anzahl Pixel für den Abstand des Fensterinhalts zum oberen und unteren Fensterrand |
| <code>marginWidth</code>  | <b>Lesen</b><br><b>Ändern</b> | Anzahl Pixel für den Abstand des Fensterinhalts zum linken und rechten Fensterrand |
| <code>name</code>         | <b>Lesen</b><br><b>Ändern</b> | Name für das Frame-Fenster                                                         |
| <code>noResize</code>     | <b>Lesen</b><br><b>Ändern</b> | Größe veränderbar oder nicht veränderbar                                           |
| <code>scrolling</code>    | <b>Lesen</b><br><b>Ändern</b> | Scrolleisen oder keine Scrolleisen                                                 |
| <code>src</code>          | <b>Lesen</b><br><b>Ändern</b> | URI für den Inhalt des Frame-Fensters                                              |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function nullRand() {
    document.getElementById("F1").marginHeight = "0";
    document.getElementById("F1").marginWidth = "0";
}
-->
</script>
</head>
<frameset cols="50%,50%">
  <frame id="F1" src="frame1.htm" onLoad="nullRand()">
  <frame id="F2" src="frame2.htm">
</frameset>
</html>
```

Das Beispiel definiert ein Frameset mit zwei Frames. Beim ersten frame-Element ist der Event-Handler `onLoad` notiert. Nach dem Laden des Frame-Fensters wird die JavaScript-Funktion `nullRand()` aufgerufen, die im Dateikopf notiert ist. Die Funktion greift mit `document.getElementById("F1")` auf das erste frame-Element zu und setzt dessen Eigenschaften für `marginHeight` und `marginWidth` auf 0.

**Beachten Sie:** Das Beispiel war weder mit Netscape noch mit dem Internet Explorer nachvollziehbar.



### frameset (HTML-Elementobjekt)

HTML-Elemente `<frameset>...</frameset>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                                                      |
|-------------|-------------------------------|----------------------------------------------------------------|
| cols        | <b>Lesen</b><br><b>Ändern</b> | Breitenverhältnisse für spaltenartig angeordnete Frame-Fenster |
| rows        | <b>Lesen</b><br><b>Ändern</b> | Höhenverhältnisse für reihenartig angeordnete Frame-Fenster    |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function dreiFenster() {
    document.getElementById("Fenster").cols = "33%,34%,33%";
    var F3 = document.createElement("frame");
    F3.setAttribute("src", "frame3.htm");
    document.getElementById("Fenster").appendChild(F3);
}
-->
```



```

</script>
</head>
<frameset id="Fenster" cols="50%,50%" onLoad="dreiFenster()">
  <frame src="frame1.htm">
  <frame src="frame2.htm">
</frameset>
</head></html>

```

Das Beispiel definiert ein Frameset mit zwei Frames. Nachdem das Frameset geladen ist (Event-Handler `onLoad` im einleitenden `<frameset>`-Tag), wird die Funktion `dreiFenster()` aufgerufen. Diese Funktion greift mit `document.getElementById("Fenster")` auf das `frameset`-Element zu und setzt dessen Eigenschaft `cols` neu – und zwar so, dass nun drei Spalten definiert werden. Dann wird mit `document.createElement()` ein neues Element erzeugt, dem mit `setAttribute()` eine anzuzeigende Datei zugewiesen wird. Anschließend wird der neue Elementknoten mit `appendChild()` als letztes neues Kindelement in das `frameset`-Element eingefügt.

**Beachten Sie:** Das Beispiel war mit dem Internet Explorer 5.0 nur teilweise nachvollziehbar.



### ***h1-h6 (HTML-Elementobjekt)***

HTML-Elemente `<h1>...</h1>` usw. haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft        | Status | Bedeutung   |
|--------------------|--------|-------------|
| <code>align</code> |        | Ausrichtung |

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function rechtslinks() {
  for(var i = 0; i < document.getElementsByTagName("h1").length; i++ ) {
    if(document.getElementsByTagName("h1")[i].align == "left")
      document.getElementsByTagName("h1")[i].align = "right";
    else
      document.getElementsByTagName("h1")[i].align = "left";
  }
}
-->
</script></head><body>
<h1 align="left">Kapitel 1</h1>
<p>viel Text</p>
<h1 align="left">Kapitel 2</h1>
<p>viel Text</p>
<h1 align="left">Kapitel 3</h1>
<p>und ein <a href="javascript:rechtslinks()">Ausrichtungsverweis</a></p>
</body></html>

```

Das Beispiel enthält insgesamt drei Überschriften erster Ordnung. Ganz unten ist ein Verweis notiert, bei dessen Anklicken die JavaScript-Funktion `rechtslinks()` aufgerufen wird, die im Dateikopf steht. Diese greift in einer `for`-Schleife der Reihe nach mit `document.getElementsByTagName("h1")[i]` auf alle `h1`-Elemente des Dokuments zu. Für jedes dieser Elemente wird abgefragt, ob seine Eigenschaft `align` den Wert `left` hat. Wenn ja, wird er auf `right` gesetzt, wenn nein, wird er wieder auf `left` gesetzt. Bei jedem Anklicken des Verweises ändern also alle drei Überschriften ihre Ausrichtung.

**Beachten Sie:** Das Beispiel ist im Netscape 6.1 nicht nachvollziehbar.



### *head* (HTML-Elementobjekt)

Das HTML-Element `<head>...</head>` hat als DOM-Objekt für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

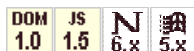
| Eigenschaft          | Status                        | Bedeutung                |
|----------------------|-------------------------------|--------------------------|
| <code>profile</code> | <b>Lesen</b><br><b>Ändern</b> | URI für Metadaten-Profil |

Beispiel:

```
<html><head profile="http://localhost/profiles/mydocs"><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write("Verwendetes Profil: " +
document.getElementsByTagName("head")[0].profile);
//-->
</script>
</body></html>
```

Das Beispiel enthält im einleitenden `<head>`-Tag eine `profile`-Angabe. Innerhalb des Dokuments wird diese Angabe mit `document.write()` ins Dokument geschrieben. Dabei wird mit `document.getElementsByTagName("head")[0]` auf das `head`-Element zugegriffen.

**Beachten Sie:** Die Verwendung anderer Universaleigenschaften als `dir` und `lang` ist im HTML 4.0-Standard für das `head`-Element nicht vorgesehen. Deshalb sollten Sie bei diesem Element die Methode `document.getElementById()` vermeiden.



### *hr* (HTML-Elementobjekt)

HTML-Elemente `<hr>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                                                            | Bedeutung                                             |
|-------------|-----------------------------------------------------------------------------------|-------------------------------------------------------|
| align       |  | Ausrichtung der Trennlinie                            |
| noShade     |  | Trennlinie wird flach und ohne 3-D-Effekt dargestellt |
| size        |  | Dicke der Trennlinie                                  |
| width       |  | Länge der Trennlinie                                  |

Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
var Dicke = 3;
function dicker() {
  Dicke += Dicke;
  document.getElementById("Linie").size = Dicke;
}
-->
</script></head><body>
<hr id="Linie" noshade size="3" onClick="dicker()">
</body></html>
```

Das Beispiel enthält eine Trennlinie der Dicke 3 (`size="3"`). Das `hr`-Element enthält einen Event-Handler `onClick`. Beim Anklicken der Trennlinie wird die JavaScript-Funktion `dicker()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion erhöht den Wert der Variablen `Dicke` um sich selbst und greift dann mit `document.getElementById("Linie")` auf das `hr`-Element zu, um dessen Eigenschaft `size` mit dem neuen Wert von `Dicke` zu belegen. Die Linie wird also bei jedem Anklicken deutlich dicker.



### html (HTML-Elementobjekt)

Das HTML-Element `<html>...</html>` hat als DOM-Objekt für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft | Status                                                                              | Bedeutung          |
|-------------|-------------------------------------------------------------------------------------|--------------------|
| version     |  | HTML-Sprachversion |

Ein Beispiel:

```
<html version="-//W3C//DTD HTML 4.01 Transitional//EN">
<head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
```

```
document.write("Dies ist in HTML " +
document.getElementsByTagName("html")[0].version + " geschrieben!");
//-->
</script>
</body></html>
```

Das Beispiel enthält im einleitenden `<html>`-Tag eine `version`-Angabe. Innerhalb des Dokuments wird diese Angabe mit `document.write()` ins Dokument geschrieben. Dabei wird mit `document.getElementsByTagName("html")[0]` auf das `html`-Element zugegriffen.

**Beachten Sie:** Die Verwendung anderer Universeigenschaften als `dir` und `lang` ist im HTML-4.0-Standard für das `html`-Element nicht vorgesehen. Deshalb sollten Sie bei diesem Element die Methode `document.getElementById()` vermeiden.



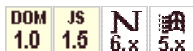
### *i* (HTML-Elementobjekt)

HTML-Elemente `<i>...</i>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universeigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">

</style>
<script type="text/javascript">
<!--
function set_normal() {
document.getElementById("schief").className = "normal";
}
//-->
</script>
</head><body>
<i id="schief">schiefe Aussage!</i><br>
<a href="javascript:set_normal()">gerade machen!</a>
</body></html>
```

Das Beispiel enthält einen mit `<i>...</i>` formatierten Text und darunter einen Verweis. Beim Anklicken des Verweises wird die JavaScript-Funktion `set_normal()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("schief")` auf das `i`-Element zu und setzt dessen Universeigenschaft `className` neu. Die entsprechende Klasse `normal` ist im Dateikopf in einem Style-Bereich definiert.



### *iframe* (HTML-Elementobjekt)

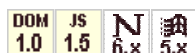
HTML-Elemente `<iframe>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universeigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft  | Status                                          | Bedeutung                                                                          |
|--------------|-------------------------------------------------|------------------------------------------------------------------------------------|
| align        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausrichtung                                                                        |
| frameBorder  | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Rahmen sichtbar oder nicht sichtbar                                                |
| height       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzeigehöhe                                                                        |
| longDesc     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | URI für längere Beschreibung zum Inhalt des Frame-Fensters                         |
| marginHeight | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzahl Pixel für den Abstand des Fensterinhalts zum oberen und unteren Fensterrand |
| marginWidth  | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzahl Pixel für den Abstand des Fensterinhalts zum linken und rechten Fensterrand |
| name         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Name für das Frame-Fenster                                                         |
| scrolling    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Scroll-Leisten oder keine Scroll-Leisten                                           |
| src          | <a href="#">Lesen</a><br><a href="#">Ändern</a> | URI für den Inhalt des Frame-Fensters                                              |
| width        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzeigebreite                                                                      |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function groesser() {
  document.getElementById("Frame").width = "600";
  document.getElementById("Frame").height = "400";
}
//-->
</script></head><body>
<iframe id="Frame" src="frame1.htm" width="400" height="300"></iframe>
<p><a href="javascript:groesser()">Anzeigefenster erweitern</a></p>
</body></html>
```

Das Beispiel definiert einen eingebetteten Frame. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die JavaScript-Funktion `groesser()` aufgerufen, die im Dateikopf notiert ist. Diese greift mit `document.getElementById("Frame")` auf das `iframe`-Element zu und ändert dessen Eigenschaften `width` und `height`. Das eingebettete Frame-Fenster wird dadurch dynamisch vergrößert.



**img (HTML-Elementobjekt)**

HTML-Elemente `<img>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universal-eigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                                                               |
|-------------|-------------------------------|-------------------------------------------------------------------------|
| align       | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung der Grafik                                                  |
| alt         | <b>Lesen</b><br><b>Ändern</b> | Alternativtext                                                          |
| border      | <b>Lesen</b><br><b>Ändern</b> | Rahmendicke um Grafik                                                   |
| height      | <b>Lesen</b><br><b>Ändern</b> | Anzeigehöhe                                                             |
| hspace      | <b>Lesen</b><br><b>Ändern</b> | horizontaler Abstand zwischen Grafik und umfließendem Text              |
| isMap       | <b>Lesen</b><br><b>Ändern</b> | Grafik benutzt ein serverseitiges Handling für verweissensitive Flächen |
| longDesc    | <b>Lesen</b><br><b>Ändern</b> | URI für Langbeschreibung zur Grafik                                     |
| lowSrc      | <b>Lesen</b><br><b>Ändern</b> | URI einer Vorabgrafik                                                   |
| name        | <b>Lesen</b><br><b>Ändern</b> | Name der Grafik                                                         |
| src         | <b>Lesen</b><br><b>Ändern</b> | URI der Grafik                                                          |
| useMap      | <b>Lesen</b><br><b>Ändern</b> | Verweis zu einem map-Bereich                                            |
| vspace      | <b>Lesen</b><br><b>Ändern</b> | vertikaler Abstand zwischen Grafik und umfließendem Text                |
| width       | <b>Lesen</b><br><b>Ändern</b> | Anzeigebreite                                                           |

### Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function neuesBild() {
if(document.getElementById("Bild").src.indexOf("iso8859_1.gif") > -1) {
  document.getElementById("Bild").src = "../../inter/iso8859_2.gif";
  document.getElementById("Bild").title = "Bild 2";
  return;
}
if(document.getElementById("Bild").src.indexOf("iso8859_2.gif") > -1) {
  document.getElementById("Bild").src = "../../inter/iso8859_3.gif";
  document.getElementById("Bild").title = "Bild 3";
  return;
}
if(document.getElementById("Bild").src.indexOf("iso8859_3.gif") > -1) {
  document.getElementById("Bild").src = "../../inter/iso8859_1.gif";
  document.getElementById("Bild").title = "Bild 1";
  return;
}
}
//-->

```

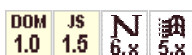
```

</script></head><body>
<br>
<a href="javascript:neuesBild()">neues Bild</a>
</body></html>

```

Das Beispiel enthält eine Grafikreferenz für eine Ressource namens *8859\_1.gif*. Unterhalb davon ist ein Verweis notiert. Bei Anklicken des Verweises wird die JavaScript-Funktion `neuesBild()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion fragt mit `document.getElementById("Bild").src.indexOf(...)` ab, welches Bild gerade angezeigt wird (siehe auch `indexOf()`). Abhängig davon werden die beiden Eigenschaften `src` und `title` mit neuen Werten belegt. Das Beispiel bewirkt, dass bei jedem Anklicken des Verweises ein neues Bild angezeigt wird.

**Beachten Sie:** In der Eigenschaft `src` ist der vollständige URI des Bildes gespeichert, und nicht die eventuell relative Angabe, die bei `<img src=>` zugewiesen wurde. Deshalb muss der Inhalt von `src` auch mit einer Zeichenkettenmethode wie `indexOf()` durchsucht werden, um nach einem Dateinamen zu suchen. In Netscape 6.x sind die Eigenschaften `hspace`, `vspace` und `border` fehlerhaft implementiert. Opera 5.12 interpretiert dieses Beispiel auch.



### input (HTML-Elementobjekt)

HTML-Elemente `<input>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universal-eigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft                 | Status          | Bedeutung                                                                 |
|-----------------------------|-----------------|---------------------------------------------------------------------------|
| <code>accept</code>         | Lesen<br>Ändern | bei <code>type="file"</code> erlaubte Mime-Typen für File-Upload          |
| <code>accessKey</code>      | Lesen<br>Ändern | Hotkey für das Element                                                    |
| <code>align</code>          | Lesen<br>Ändern | Ausrichtung                                                               |
| <code>alt</code>            | Lesen<br>Ändern | Alternativtext bei <code>type="image"</code>                              |
| <code>checked</code>        | Lesen<br>Ändern | Auswählen bei <code>type="radio"</code> oder <code>type="checkbox"</code> |
| <code>defaultValue</code>   | Lesen<br>Ändern | vorbelegter Wert                                                          |
| <code>defaultChecked</code> | Lesen<br>Ändern | per Vorbelegung ausgewählt oder nicht                                     |
| <code>disabled</code>       | Lesen<br>Ändern | Element kann nicht geändert werden                                        |
| <code>form</code>           | Lesen           | zugehöriges Formular                                                      |
| <code>maxLength</code>      | Lesen<br>Ändern | maximale Anzahl eingebbarer Zeichen                                       |
| <code>name</code>           | Lesen<br>Ändern | Name des Elements                                                         |
| <code>readOnly</code>       | Lesen<br>Ändern | Wert des Elements kann nicht verändert werden                             |
| <code>size</code>           | Lesen<br>Ändern | Anzeigebreite bei <code>type="text"</code>                                |

| Eigenschaft | Status                        | Bedeutung                                     |
|-------------|-------------------------------|-----------------------------------------------|
| src         | <b>Lesen</b><br><b>Ändern</b> | URI der Grafik bei type="image"               |
| tabIndex    | <b>Lesen</b><br><b>Ändern</b> | Tabulatorreihenfolge                          |
| type        | <b>Lesen</b>                  | Typ des Formularelements                      |
| useMap      | <b>Lesen</b><br><b>Ändern</b> | Verweis zu einem map-Bereich bei type="image" |
| value       | <b>Lesen</b><br><b>Ändern</b> | Wert des Elements                             |

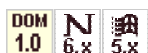
| Methode  | Bedeutung                   |
|----------|-----------------------------|
| blur()   | Fokus vom Element entfernen |
| focus()  | Fokus auf Element setzen    |
| select() | Wert selektieren            |
| click()  | Element anklicken           |

### Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Breite = 3;
function erweitern() {
    Breite += 1;
    document.getElementsByName("Eingabe")[0].size = Breite;
}
//-->
</script></head><body>
<form name="Formular">
Geben Sie Text ein:<br>
<input size="3" name="Eingabe" onKeyPress="erweitern()">
</form>
</body></html>
```

In der Beispieldatei ist ein Formular mit einem zunächst ziemlich kleinen Eingabefeld (size="3") notiert. Das Eingabefeld enthält den Event-Handler onKeyPress. Bei jedem Tastendruck, der erfolgt, während das Eingabefeld den Fokus hat, wird darum die JavaScript-Funktion erweitern() aufgerufen, die im Dateikopf notiert ist. Diese greift mit document.getElementsByName("Eingabe")[0] auf das erste Element mit dem Namen Eingabe zu, also das Eingabefeld, und ändert dessen Eigenschaft size mit dem aktuellen Wert der Variablen Breite. Diese wird bei jedem Aufruf der Funktion um 1 erhöht. Das Beispiel bewirkt, dass das Eingabefeld um so größer wird, je mehr Text eingegeben wird.



**ins (HTML-Elementobjekt)**

HTML-Elemente `<ins>...</ins>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

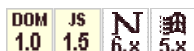
| Eigenschaft | Status                        | Bedeutung                      |
|-------------|-------------------------------|--------------------------------|
| cite        | <b>Lesen</b><br><b>Ändern</b> | URI für Gründe der Löschung    |
| dateTime    | <b>Lesen</b><br><b>Ändern</b> | Datum und Uhrzeit der Löschung |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p>Text <ins id="hinzugekommen"
onMouseOver="document.getElementById('hinzugekommen').title='das ist
hinzugekommener Text'">mit weiterem Text</ins></p>
</body></html>
```

Das Beispiel enthält einen Absatz, von dessen Text ein Teil mit `<ins>...</ins>` ausgezeichnet ist. Beim Überfahren dieses Textes mit der Maus (`onMouseOver`) wird dynamisch die Universaleigenschaft `title` mit einem Wert belegt. Dazu wird mit `document.getElementById('hinzugekommen')` auf das `ins`-Element zugegriffen.

**Beachten Sie:** Mit Netscape 6.1 war dieses Beispiel nicht nachvollziehbar. Das Beispiel wird auch von Opera 5.12 interpretiert.

**isindex (HTML-Elementobjekt)**

Das HTML-Element `<isindex>` hat als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

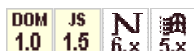
| Eigenschaft | Status                        | Bedeutung            |
|-------------|-------------------------------|----------------------|
| form        | <b>Lesen</b>                  | zugehöriges Formular |
| prompt      | <b>Lesen</b><br><b>Ändern</b> | Labeltext            |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<isindex title="selten angewendetes HTML-Element" id="Eingabe"
onMouseOver="alert(document.getElementById('Eingabe').title)">
</body></html>
```

Das Beispiel enthält ein `isindex`-Element. Beim Überfahren des Elements mit der Maus wird ein Meldungsfenster ausgegeben, das den Wert des `title`-Attributs ausgibt, das im Element notiert ist. Dazu wird mit `document.getElementById('Eingabe')` auf das Element zugegriffen.

**Beachten Sie:** Der Event-Handler `onMouseOver` kann zwar auf dieses Element angewendet werden, ist jedoch kein Bestandteil der Standards. Das vorliegende Beispiel wird auch von Opera 5.12, jedoch nicht vom MS Internet Explorer 5.0 Macintosh Edition interpretiert.



### ***kbd (HTML-Elementobjekt)***

HTML-Elemente `<kbd>...</kbd>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Hinweis() {
  document.getElementById("Keyboard").title = "Sie haben es geschafft!";
  alert(document.getElementById("Keyboard").title);
}
//-->
</script></head><body onKeyPress="Hinweis()">
<kbd id="Keyboard">Taste drücken</kbd>:
</body></html>
```

Das Beispiel enthält die Aufforderung eine Taste zu drücken, die sinnigerweise in `<kbd>...</kbd>` eingeschlossen ist. Das einleitende `<body>`-Tag der Datei enthält den Event-Handler `onKeyPress`. Wenn das Dokument den Fokus hat und der Anwender irgendeine Taste drückt, wird die JavaScript-Funktion `Hinweis()` aufgerufen, die im Dateikopf definiert ist. Diese Funktion greift mit `document.getElementById("Keyboard")` auf das `kbd`-Element zu und weist dessen Universaleigenschaft `title` einen Wert zu. Anschließend wird zur Kontrolle genau diese Universaleigenschaft in einem Meldungsfenster ausgegeben.

**Beachten Sie:** Das Beispiel wird auch von Opera 5.12 interpretiert.



### ***label (HTML-Elementobjekt)***

HTML-Elemente `<label>...</label>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft            | Status                        | Bedeutung                                |
|------------------------|-------------------------------|------------------------------------------|
| <code>accessKey</code> | <b>Lesen</b><br><b>Ändern</b> | Hotkey für das zugehörige Element        |
| <code>form</code>      | <b>Lesen</b>                  | zugehöriges Formular                     |
| <code>htmlFor</code>   | <b>Lesen</b><br><b>Ändern</b> | Id des Elements, zu dem das Label gehört |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Formular" action="">
<table>
  <tr>
    <td><label id="VN" for="Vorname">Ihr Vorname:</label></td>
    <td><input type="text" id="Vorname"></td>
  </tr><tr>
    <td><label id="ZN" for="Zuname">Ihr Zuname:</label></td>
    <td><input type="text" id="Zuname"></td>
  </tr>
</table>
</form>
<script type="text/javascript">
<!--
document.getElementById("Vorname").value = document.getElementById("VN").htmlFor;
document.getElementById("Zuname").value = document.getElementById("ZN").htmlFor;
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular mit zwei Eingabefeldern. Die Beschriftungen der Eingabefelder sind jeweils mit einem `label`-Element ausgezeichnet. Unterhalb des Formulars ist ein Script-Bereich notiert. Dort werden die beiden Eingabefelder mit Werten vorbelegt, und zwar mit den Werten, die den `id`-Attributen der zugehörigen `label`-Elemente zugewiesen sind. Dazu wird mit `document.getElementById("VN")` und `document.getElementById("ZN")` auf die `label`-Elemente zugegriffen. Ihre Eigenschaft `htmlFor` wird den Formularfeldern zugewiesen, auf die ebenfalls mit `getElementById()` zugegriffen wird.



### legend (HTML-Elementobjekt)

HTML-Elemente `<legend>...</legend>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung              |
|-------------|-------------------------------|------------------------|
| accessKey   | <b>Lesen</b><br><b>Ändern</b> | Hotkey für das Element |
| align       | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung            |
| form        | <b>Lesen</b>                  | zugehöriges Formular   |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Formular" action="">
<fieldset>
<legend id="L1">Absender</legend>
```

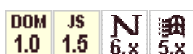
```

<input type="text" size="50" name="Absendername"
      onFocus="document.getElementById('L1').align='right'"
      onBlur="document.getElementById('L1').align='left'">
</fieldset>
<fieldset>
<legend id="L2">Mitteilung</legend>
<input type="text" size="50" name="Mitteilungstext"
      onFocus="document.getElementById('L2').align='right'"
      onBlur="document.getElementById('L2').align='left'">
</fieldset>
</form>
</body></html>

```

Das Beispiel enthält ein Formular, bei dem für jedes der beiden Eingabefelder ein `fieldset`-Bereich mitsamt `legend`-Element definiert ist. Sobald eines der Eingabefelder den Fokus erhält (`onFocus`), wird der zugehörige Legendentext rechtsbündig ausgerichtet. Sobald das Eingabefeld den Fokus wieder verliert (`onBlur`), wird der Legendentext wieder linksbündig ausgerichtet. Dazu wird mit `document.getElementById('L1')` bzw. `document.getElementById('L2')` auf die `legend`-Elemente zugegriffen. Neu gesetzt wird jeweils ihre Eigenschaft `align`.

**Beachten Sie:** Im Netscape 6 erfolgt die Ausrichtung ausgehend vom rechten Rand des Eingabefeldes. Im MS Internet Explorer 5.0 war dieses Beispiel nicht nachvollziehbar.



### li (HTML-Elementobjekt)

HTML-Elemente `<li>...</li>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                                    |
|-------------|-------------------------------|----------------------------------------------|
| type        | <b>Lesen</b><br><b>Ändern</b> | Aufzählungszeichen oder der Nummerierungstyp |
| value       | <b>Lesen</b><br><b>Ändern</b> | Wert der laufenden Nummerierung neu setzen   |

Ein Beispiel:

```

<html><head><title>Test</title>
<style type="text/css">
<!--
.hinterlegt { background-color:yellow }
-->
</style>
<script type="text/javascript">
<!--
function Aendern() {
  for(var i = 0; i < document.getElementsByTagName("li").length; i++)
    if(i % 2 != 0) {
      document.getElementsByTagName("li")[i].className = "hinterlegt";
      document.getElementsByTagName("li")[i].type = "I";
    }
}

```

```

}
//-->
</script>
</head><body>
<ol>
<li>Ein ungerader Listenpunkt</li>
<li>Ein gerader Listenpunkt</li>
<li>Ein ungerader Listenpunkt</li>
<li>Ein gerader Listenpunkt</li>
<li>Ein ungerader Listenpunkt</li>
<li>Ein gerader Listenpunkt</li>
<li>Ein ungerader Listenpunkt</li>
<li>Ein gerader Listenpunkt</li>
</ol>
<a href="javascript:Aendern()">gerade Listenpunkte hinterlegen!</a>
</body></html>

```

Das Beispiel enthält eine nummerierte Liste mit einigen Listenpunkten. Unterhalb davon ist ein Verweis notiert. Bei Anklicken des Verweises wird die JavaScript-Funktion `Aendern()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion geht in einer `for`-Schleife der Reihe nach alle `li`-Elemente durch, wobei sie mit `document.getElementsByTagName("li")` darauf zugreift. Bei geraden Listenpunkten werden zwei Änderungen durchgeführt: Erstens erhalten diese Listenpunkte eine oben im Style-Bereich definierte CSS-Klasse zugewiesen, die dem Element eine andere Hintergrundfarbe zuweist, und zweitens wird der Nummerierungstyp dieser Listenpunkte auf »römisch großgeschrieben« gesetzt.



### link (HTML-Elementobjekt)

HTML-Elemente `<link>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universal-eigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                          | Bedeutung                              |
|-------------|-------------------------------------------------|----------------------------------------|
| charset     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Zeichensatz des Verweisziels           |
| disabled    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | als Button anklickbar oder nicht       |
| href        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweisziel                            |
| hreflang    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Landessprache des Verweisziels         |
| media       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausgabemedium                          |
| rel         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweisziel als »Vorwärtsverknüpfung«  |
| rev         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweisziel als »Rückwärtsverknüpfung« |
| target      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Fenstername des Verweisziels           |
| type        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type des Verweisziels             |

Ein Beispiel:

```
<html><head><title>Test</title>
<link id="CSS" rel="stylesheet" href="styles_1.css">
<script type="text/javascript">
<!--
function CSS_Wechsel() {
  document.getElementById("CSS").href = "styles_2.css";
}
//-->
</script>
</head><body>
<h1>Dies und das</h1>
<a href="javascript:CSS_Wechsel()">anderes Stylesheet!</a>
</body></html>
```

Das Beispiel enthält im Dateikopf ein HTML-Tag zum Einbinden eines CSS-Stylesheets. Innerhalb des sichtbaren Dokumentbereichs ist ein Verweis notiert, bei dessen Anklicken die JavaScript-Funktion `CSS_Wechsel()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("CSS")` auf das `link`-Element zu, das die CSS-Datei einbindet, und weist der Eigenschaft `href` einen neuen Wert zu, nämlich die neue CSS-Datei. Dadurch wird die neue CSS-Datei relevant für alle Formatierungen innerhalb der HTML-Datei.

**Beachten Sie:** Mit Netscape 6.1 und dem MS Internet Explorer 5.0 Macintosh Edition war dieses Beispiel nicht nachvollziehbar.



### *map (HTML-Elementobjekt)*

HTML-Elemente `<map>...</map>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                         |
|-------------|-------------------------------|-----------------------------------|
| areas       | <b>Lesen</b><br><b>Ändern</b> | Array der area-Definitionen       |
| name        | <b>Lesen</b><br><b>Ändern</b> | Ankername des Definitionsbereichs |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function zeigCoords() {
  alert(document.getElementsByName("Testbild")[0].areas[0].coords);
}
//-->
</script>
</head><body>
<map name="Testbild">
```

```

<area id="Verweis" shape="rect" coords="1,1,249,49"
      href="javascript:zeigCoords()" alt="Verweis">
</map>

</body></html>

```

Das Beispiel enthält eine Grafik, die auf einen ebenfalls enthaltenen verweissensitiven Bereich verweist. Dort wird ein verweissensitives Rechteck für die Grafik definiert. Beim Anklicken dieses Bereichs wird die JavaScript-Funktion `zeigCoords()` aufgerufen, die im Dateikopf notiert ist. Die Funktion gibt die Koordinaten des verweissensitiven Bereichs aus, indem sie mit `document.getElementsByName("Testbild")[0].areas[0].coords` auf den `areas`-Array des `map`-Elements zugreift und die Eigenschaft `coords` von dessen erstem Element anspricht.

#### **DOM 1.0** *menu (HTML-Elementobjekt)*

HTML-Elemente `<menu>...</menu>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

| Eigenschaft | Status                        | Bedeutung                    |
|-------------|-------------------------------|------------------------------|
| compact     | <b>Lesen</b><br><b>Ändern</b> | Darstellung in enger Schrift |

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function zeigKompakt() {
  document.getElementsByTagName("menu")[0].compact = true;
}
//-->
</script>
</head><body>
<menu onMouseOver="zeigKompakt()">
<li>Datei</li>
<li>Bearbeiten</li>
<li>Ansicht</li>
<li>Format</li>
</menu>
</body></html>

```

Das Beispiel enthält eine Menüliste. Beim Überfahren der Liste mit der Maus (`onMouseOver`) wird die JavaScript-Funktion `zeigKompakt()` aufgerufen, die im Dateikopf notiert ist. Die Funktion greift mit `document.getElementsByTagName("menu")[0]` auf das erste `menu`-Element in der Datei zu und setzt seine Eigenschaft `compact` auf den Wert `true` (»wahr«). Dadurch wird die gesamte Liste in kompakter Schrift dargestellt.

**Beachten Sie:** Weder Netscape noch Internet Explorer interpretieren das HTML-Attribut compact.

### **DOM 1.0** meta (HTML-Elementobjekt)

HTML-Elemente `<meta>...</meta>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                        | Bedeutung                                               |
|-------------|-------------------------------|---------------------------------------------------------|
| content     | <b>Lesen</b><br><b>Ändern</b> | Wert/Inhalt der Meta-Angabe                             |
| httpEquiv   | <b>Lesen</b><br><b>Ändern</b> | Eigenschaft für Meta-Informationen (für HTTP-Kopfdaten) |
| name        | <b>Lesen</b><br><b>Ändern</b> | Eigenschaft für Meta-Informationen                      |
| schema      | <b>Lesen</b><br><b>Ändern</b> | Schema für den Wert der Eigenschaft                     |

Ein Beispiel:

```
<html><head><title>Test</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<script type="text/javascript">
<!--
function Arabisch() {
  document.getElementsByTagName("meta")[0].content="text/html; charset=iso-8859-6";
}
//-->
</script>
</head><body>
<p style="font-size:24pt">&#200;&#201;&#202;&#203;&#204;&#205;&#206;&#207;&#208;&#209;</p>
<p><a href="javascript:Arabisch()">Arabisch</a></p>
</body></html>
```

Das Beispiel enthält einen Textabsatz, der mit numerischen Entities Zeichenwerte von 200 bis 209 erzeugt. Unterhalb davon ist ein Verweis notiert. Beim Anklicken wird die JavaScript-Funktion `Arabisch()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementsByTagName("meta")[0]` auf das meta-Tag zu, das ebenfalls im Dateikopf notiert ist, und ändert dessen Eigenschaft `content` so, dass der arabische Zeichensatz (iso-8859-6) angegeben wird.

**Beachten Sie:** Weder Netscape noch Internet Explorer interpretieren diese Veränderung.

### **DOM 1.0** noframes (HTML-Elementobjekt)

HTML-Elemente `<noframes>...</noframes>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften.



Ein Beispiel:

```
<html><head><title>Test</title>
</head>
<frameset cols="50%,50%">
  <frame id="F1" src="frame1.htm">
  <frame id="F2" src="frame2.htm">
  <noframes title="Das ist aber ein doofer Browser!">
    Ihr Browser kann keine Frames anzeigen!
    <a
href="javascript:alert(document.getElementsByTagName('noframes')[0].title)">weitere
Infos</a>
  </noframes>
</frameset>
</html>
```

Das Beispiel enthält eine Frameset-Definition und einen alternativen `noframes`-Bereich. Innerhalb davon ist ein Verweis notiert, bei dessen Anklicken der Wert der Universaleigenschaft `title` des einleitenden `<noframes>`-Tags in einem Meldungsfenster ausgegeben wird.

**Beachten Sie:** Leider ist dieses Beispiel mit den gängigen Browsern nicht nachvollziehbar, da diese Frames unterstützen und auch kein Abschalten von Frames ermöglichen. Eine Ausnahme bildet hier Opera, der ein Abschalten von Frames zulässt und das Beispiel auch interpretiert.

#### **DOM 1.0** *noscript (HTML-Elementobjekt)*

HTML-Elemente `<noscript>...</noscript>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
</head><body>
<script language="tcl" type="text/tcl">
<!--
  proc square {i} {
    document write "The call passed $i to the function.<BR>"
    return [expr $i * $i]
  }
  document write "The function returned [square 5]."
```

```
# -->
</script>
<noscript id="tcl" title="Ich verstehe kein tcl.">
  <p onmouseover="alert(document.getElementById('tcl').title)">Ein Browser der kein
tcl kann.</p>
</noscript>
</body></html>
```

Im Beispiel wurde ein Script-Bereich in der Sprache `tcl` dargestellt. Browser, die diese Sprache nicht interpretieren, interpretieren den zugehörigen `noscript`-Bereich und zeigen den alternativen Text an. Da es sich um eine andere Sprache als JavaScript handelt, können Sie auf das Scriptelement mit JavaScript zugreifen. Wenn die Maus über den Absatz bewegt wird, wird der Titel des `noscript`-Elementes angezeigt.

**Beachten Sie:** Der MS Internet Explorer und Netscape interpretieren diesen Tag nur bei ausgeschalteten JavaScript. In Opera 5.12 können Sie das Beispiel jedoch nachvollziehen.



### *object (HTML-Elementobjekt)*

HTML-Elemente `<object>...</object>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                          | Bedeutung                                                           |
|-------------|-------------------------------------------------|---------------------------------------------------------------------|
| align       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausrichtung des Objekts                                             |
| archive     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | komma-separierte Liste mit Archivdateien für das Objekt             |
| border      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Rahmendicke um Objekt                                               |
| code        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Klassendatei bei Java-Applets                                       |
| codeBase    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Basis-URI für Angaben zu <code>data</code> und <code>archive</code> |
| codeType    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type der Datenquelle                                           |
| data        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | URI der Datenquelle                                                 |
| declare     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Objekt wird geladen, aber nicht initialisiert                       |
| form        | <a href="#">Lesen</a>                           | zugehöriges Formularelement                                         |
| height      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzeigehöhe                                                         |
| hspace      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | horizontaler Abstand zwischen Objekt und umfließendem Text          |
| name        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Name der Grafik                                                     |
| standBy     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Meldungstext, der angezeigt wird, während das Objekt geladen wird   |
| tabIndex    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Tabulatorreihenfolge für Objekte                                    |
| type        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type der Datenquelle                                           |
| useMap      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Verweis zu einem <code>map</code> -Bereich                          |
| vspace      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | vertikaler Abstand zwischen Objekt und umfließendem Text            |
| width       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzeigebreite                                                       |

Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
function groesser() {
```

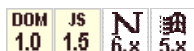
```

document.getElementById("TXT").width="640";
document.getElementById("TXT").height="480";
}
//-->
</script>
</head><body>
<p><object id="TXT" data="html.txt" type="text/plain" width="320"
height="240"></object></p>
<p><a href="javascript:groesser()">mehr sehen!</a></p>
</body></html>

```

Das Beispiel enthält die Objektdefinition einer TXT-Datei. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die JavaScript-Funktion `groesser()` aufgerufen, die im Dateikopf notiert ist. Die Funktion greift mit `document.getElementById("TXT")` auf das object-Element zu und ändert dessen Eigenschaften für `width` und `height`. Das Objekt wird dadurch dynamisch vergrößert.

**Beachten Sie:** Im Netscape 6.1 war dieses Beispiel nicht nachvollziehbar.



### ol (HTML-Elementobjekt)

HTML-Elemente `<ol>...</ol>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                          | Bedeutung                      |
|-------------|-------------------------------------------------|--------------------------------|
| compact     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Darstellung in enger Schrift   |
| start       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Startwert für die Nummerierung |
| type        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Art der Nummerierung           |

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function roemisch() {
  document.getElementById("Liste").type="I";
}
function normal() {
  document.getElementById("Liste").type="1";
}
//-->
</script>
</head><body>
<ol id="Liste">
<li>eins</li>
<li>zwei</li>

```

```

<li>drei</li>
<li>vier</li>
<li>fünf</li>
</ol>
<a href="javascript:roemisch()">römisch</a><br>
<a href="javascript:normal()">normal</a>
</body></html>

```

Das Beispiel enthält eine nummerierte Liste. Unterhalb davon sind zwei Verweise notiert. Der eine ruft eine JavaScript-Funktion `roemisch()` auf, der andere eine Funktion `normal()`. Beide Funktionen greifen jeweils mit `document.getElementById("Liste")` auf das Start-Tag der nummerierten Liste zu und ändern dessen Eigenschaft `type`. Im einen Fall wird die Nummerierung dynamisch durch römische Ziffern ersetzt, im andern Fall werden wieder arabische Ziffern benutzt.



### **optgroup (HTML-Elementobjekt)**

HTML-Elemente `<optgroup>...</optgroup>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft           | Status                        | Bedeutung                                         |
|-----------------------|-------------------------------|---------------------------------------------------|
| <code>disabled</code> | <b>Lesen</b><br><b>Ändern</b> | Gruppe von Einträgen kann nicht ausgewählt werden |
| <code>label</code>    | <b>Lesen</b><br><b>Ändern</b> | Text des Gruppeneintrags                          |

Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<form name="Auswahl" action="">
<select name="Namen" size="1" onChange="alert(document.getElementById('A').label)">
  <optgroup label="Namen mit A" id="A">
    <option label="Anna"> Anna</option>
    <option label="Anke"> Anke</option>
  </optgroup>
</select>
</form>
</body></html>

```

Das Beispiel definiert eine Menüstruktur. Sobald ein Eintrag ausgewählt ist (`onChange`), wird in einem Meldungsfenster der Wert der Eigenschaft `label` des `optgroup`-Elements mit dem `id`-Wert A ausgegeben.

**Beachten Sie:** Weder Netscape noch Internet Explorer interpretieren Menüstrukturen bis auf den heutigen Tag korrekt – Netscape 6.x zwar ansatzweise, aber noch nicht so wie es gedacht ist. Dennoch interpretieren beide Browser das obige JavaScript.

**option (HTML-Elementobjekt)**

HTML-Elemente `<option>...</option>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft     | Status                        | Bedeutung                                                                                  |
|-----------------|-------------------------------|--------------------------------------------------------------------------------------------|
| defaultSelected | <b>Lesen</b><br><b>Ändern</b> | vorausgewählter Auswahllisteneintrag                                                       |
| disabled        | <b>Lesen</b><br><b>Ändern</b> | Auswahllisteneintrag kann nicht ausgewählt werden                                          |
| form            | <b>Lesen</b>                  | zugehöriges Formular                                                                       |
| index           | <b>Lesen</b>                  | Indexnummer des Auswahllisteneintrags (erster Eintrag hat Nummer 0, zweiter Nummer 1 usw.) |
| label           | <b>Lesen</b><br><b>Ändern</b> | Text des Eintrags im Zusammenhang mit Menüstrukturen                                       |
| selected        | <b>Lesen</b><br><b>Ändern</b> | Auswahllisteneintrag wird ausgewählt                                                       |
| text            | <b>Lesen</b>                  | Text des Auswahllisteneintrags                                                             |
| value           | <b>Lesen</b><br><b>Ändern</b> | Absendewert des Auswahllisteneintrags                                                      |

**Ein Beispiel:**

```

<html><head><title>Test</title>
</head><body>
<form name="Auswahl" action="">
<select name="Zutaten" size="1">
  <option value="Z_101">Salami</option>
  <option value="Z_102">Pilze</option>
  <option value="Z_103">Schinken</option>
  <option value="Z_104">Oliven</option>
  <option value="Z_105">Paprika</option>
</select>
</form>
<script language="JavaScript" type="text/javascript">
<!--
document.write("<table border=\\"1\\">");

for(i = 0; i < document.getElementsByTagName("option").length; i++) {
  document.write("<tr><td><b>Zutat:</b></td><td>");
  document.write(document.getElementsByTagName("option")[i].text);
  document.write("</td><td><b>interner Absendewert:</b></td><td>");
  document.write(document.getElementsByTagName("option")[i].value);
  document.write("</td></tr>");
}

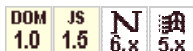
document.write("</table>");

```

```
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular mit einer Auswahlliste. Unterhalb davon ist ein Script notiert, das dynamisch eine Tabelle ins Dokument schreibt. Dazu geht es in einer `for`-Schleife der Reihe nach alle `option`-Elemente durch und greift mit `document.getElementsByTagName("option")` auf die einzelnen `option`-Elemente zu. Für jedes Element werden der interne Absendewert (`value`) und der Auswahltext (`text`) in die Tabelle geschrieben.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



### *p* (HTML-Elementobjekt)

HTML-Elemente `<p>...</p>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.

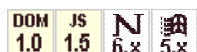
| Eigenschaft        | Status | Bedeutung   |
|--------------------|--------|-------------|
| <code>align</code> |        | Ausrichtung |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Ausrichten(wie) {
  for(var i = 0; i < document.getElementsByTagName("p").length; i++) {
    document.getElementsByTagName("p")[i].align = wie;
  }
}
//-->
</script>
</head><body>
<p>Der erste</p>
<p>Der zweite</p>
<p>Der dritte</p>
<form name="Formular" action="">
<input type="button" value="links" onClick="Ausrichten('left')">
<input type="button" value="zentriert" onClick="Ausrichten('center')">
<input type="button" value="rechts" onClick="Ausrichten('right')">
</form>
</body></html>
```

Das Beispiel enthält drei Textabsätze. Unterhalb davon ist ein Formular mit drei Klick-Buttons notiert. Jeder Button ruft beim Anklicken die JavaScript-Funktion `Ausrichten()` auf, die im Dateikopf notiert ist, und übergibt ihr einen Wunschwert für die Ausrichtung (`left`, `center` und `right`). Die Funktion `Ausrichten()` geht in einer `for`-Schleife mit `document.getEle-`

mentsByTagName("p") der Reihe nach alle p-Elemente des Dokuments durch und weist ihren Eigenschaften align den übergebenen Parameter für die Ausrichtung zu. Dadurch bewirkt der Klick auf einen der Buttons die dynamische Ausrichtung aller Textabsätze.



### **param (HTML-Elementobjekt)**

HTML-Elemente `<param>...</param>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                          | Bedeutung                              |
|-------------|-------------------------------------------------|----------------------------------------|
| name        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Name des Parameters                    |
| type        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type des Wertes                   |
| value       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Initialisierungswert für den Parameter |
| valueType   | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Informationen zum Attribut type        |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<applet id="Ticker" code="zticker.class" width="600" height="60">
  <param name="msg" value="Die Energie des Verstehens">
  <param name="speed" value="5">
  <param name="bgco" value="255,255,255">
  <param name="txtco" value="000,000,255">
  <param name="hrefco" value="255,255,255">
</applet>
<script type="text/javascript">
<!--
document.write("<table>");
for(var i = 0; i < document.getElementsByTagName("param").length; i++) {
  document.write("<tr><td><b>Parameter:</b></td><td>");
  document.write(document.getElementsByTagName("param")[i].name);
  document.write("</td><td><b>Wert:</b></td><td>");
  document.write(document.getElementsByTagName("param")[i].value);
  document.write("</td></tr>");
}
document.write("</table>");
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Java-Applet mit verschiedenen Parametern. Unterhalb davon ist ein Script notiert, das dynamisch eine Tabelle ins Dokument schreibt. Dazu geht es in einer for-Schleife der Reihe nach alle param-Elemente durch und greift mit document.getElements-

ByTagName("param") auf die einzelnen param-Elemente zu. Für jedes Element werden der Parametername (name) und der zugehörige Wert (value) in die Tabelle geschrieben.

**Beachten Sie:** Im MS Internet Explorer bis Version 5.5 lässt sich dieses Beispiel merkwürdigerweise nur nachvollziehen, wenn sich die param-Elemente nicht innerhalb eines gültigen Eltern-elementes befinden. Das vorliegende Beispiel wird dagegen auch von Opera 5.12 und dem MS Internet Explorer 5.0 Macintosh Edition interpretiert.



### pre (HTML-Elementobjekt)

HTML-Elemente <pre>...</pre> haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status | Bedeutung               |
|-------------|--------|-------------------------|
| width       |        | feste Breite für Inhalt |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Breite() {
  document.getElementById("x").width = "15";
}
//-->
</script>
</head><body>
<pre id="x">
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
</pre>
<a href="javascript:Breite()">Breite auf 15 setzen</a>
</body></html>
```

Das Beispiel enthält einen pre-Bereich mit 30 x-Zeichen pro Zeile. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die JavaScript-Funktion Breite() aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit document.getElementById("x") auf das pre-Element zu und weist ihm für die Eigenschaft width den Wert 15 zu.

**Beachten Sie:** Weder Netscape noch Internet Explorer interpretieren das Attribut width im Zusammenhang mit pre-Elementen.



### q (HTML-Elementobjekt)

HTML-Elemente <q>...</q> haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigene Eigenschaft.



| Eigenschaft | Status                                                                            | Bedeutung                     |
|-------------|-----------------------------------------------------------------------------------|-------------------------------|
| cite        |  | URI für die Quelle des Zitats |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p>Wie sagte Goethe doch?
<q id="Faustzitat" cite="aus Faust"
onMouseOver="alert(document.getElementById('Faustzitat').cite)">Habe, ach,
Philosophie studiert ...</q></p>
</body></html>
```

Das Beispiel enthält ein Zitat, in dessen einleitendem Tag ein Event-Handler `onMouseOver` notiert ist. Beim Überfahren mit der Maus wird ein Meldungsfenster ausgegeben, das den Wert des Attributs `cite=` ausgibt. Dazu wird mit `document.getElementById('Faustzitat').cite` auf die entsprechende Elementeigenschaft zugegriffen.

**Beachten Sie:** Netscape 6, Opera 5.12 und der MS Internet Explorer 5.0 Macintosh Edition schließen das Zitat in `"..."` ein. Opera 5.12 interpretiert zwar den Event-Handler, gibt aber als Eigenschaft des Attributes `cite=` den Wert `undefined` zurück.



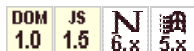
### s (HTML-Elementobjekt)

HTML-Elemente `<s>...</s>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.rot { color:red }
-->
</style>
<script type="text/javascript">
<!--
function deutlicher() {
  document.getElementById("schweinisch").className = "rot";
}
//-->
</script>
</head><body>
<h1><s id="schweinisch">Es tut so gut ein Schwein zu sein</s></h1>
<a href="javascript:deutlicher()">deutlicher!</a>
</body></html>
```

Das Beispiel enthält eine Überschrift mit durchgestrichenem Text. Unterhalb davon steht ein Verweis, bei dessen Anklicken die JavaScript-Funktion `deutlicher()` aufgerufen wird. Mit `document.getElementById("schweinisch")` greift diese Funktion auf das `s`-Element zu und weist ihm für die Universaleigenschaft `className` den Klassennamen `rot` zu, für den im Datei-

kopf in einem Style-Bereich CSS-Definitionen existieren. Der durchgestrichene Text wird dadurch rot.



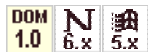
### **samp (HTML-Elementobjekt)**

HTML-Elemente `<samp>...</samp>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p>Ich sage zum Beispiel immer <samp id="zumbeispiel"
onMouseOver="alert(document.getElementById('zumbeispiel').id)">zum
Beispiel</samp></p>
</body></html>
```

Das Beispiel enthält einen mit `<samp>...</samp>` ausgezeichneten Text. Im Einleitungs-Tag ist der Event-Handler `onMouseOver` notiert. Beim Überfahren des Textes mit der Maus wird in einem Meldungsfenster der Id-Wert des Elements ausgegeben. Dazu wird mit `document.getElementById('zumbeispiel').id` auf die entsprechende Eigenschaft zugegriffen.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



### **script (HTML-Elementobjekt)**

HTML-Elemente `<script>...</script>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status          | Bedeutung                                                         |
|-------------|-----------------|-------------------------------------------------------------------|
| charset     | Lesen<br>Ändern | Zeichensatz der bei <code>src</code> angegebenen Script-Datei     |
| defer       | Lesen<br>Ändern | Script kann keine Inhalte im Dokumentfenster ändern oder erzeugen |
| event       | Lesen<br>Ändern | Ereignis für Script-Ausführung                                    |
| htmlFor     | Lesen<br>Ändern | Element, für das das Script ausgeführt wird                       |
| src         | Lesen<br>Ändern | URI einer Script-Datei                                            |
| text        | Lesen<br>Ändern | Inhalt des Scripts                                                |
| type        | Lesen<br>Ändern | Mime-Type der Script-Sprache                                      |

Beispiel: JavaScript-Datei *hallo\_deutsch.js*:

```
function hallo()
{
  alert("Hallo Welt");
}
```

Beispiel: JavaScript-Datei *hallo\_englisch.js*:

```
function hallo()
{
  alert("hello world");
}
```

Beispiel: HTML-Datei

```
<html><head><title>Test</title>
<script src="hallo_deutsch.js" type="text/javascript">
</script>
<script type="text/javascript">
<!--
  function englisch() {
    if (document.getElementsByTagName("script")[0].src!="hallo_englisch.js")
      document.getElementsByTagName("script")[0].src="hallo_englisch.js";
    hallo();
  }
  function deutsch() {
    if (document.getElementsByTagName("script")[0].src!="hallo_deutsch.js")
      document.getElementsByTagName("script")[0].src="hallo_deutsch.js";
    hallo();
  }

  //-->
</script>
</head><body>
<a href="javascript:englisch()">englisches Hallo-Welt</a><br>
<a href="javascript:deutsch()">deutsches Hallo-Welt</a>
</body></html>
```

Für dieses Beispiel wurden zwei externe JavaScript-Dateien definiert. Beide Dateien enthalten die gleiche Funktion `hallo()`. Sie unterscheiden sich jedoch darin, dass in der Datei *hallo\_deutsch.js*, die Ausgabe in Deutsch erfolgt und in der Datei *hallo\_englisch.js* in Englisch.

Im Beispiel gibt es zwei Script-Bereiche. Im ersten Bereich wird unter Verwendung der `src`-Eigenschaft das externe JavaScript *hallo\_deutsch.js* geladen. Im zweiten Bereich sind die Funktionen `englisch()` und `deutsch()` notiert. Der Verweis *englisches Hallo-Welt* ruft die Funktion `englisch()` auf. Innerhalb der Funktion wird geprüft, ob dem ersten `script`-Element die Datei *hallo\_englisch.js* als `src`-Eigenschaft zugewiesen ist. Ist dies nicht der Fall, so erhält dieses Element die Datei *hallo\_englisch.js* zugewiesen und die Datei wird geladen. Die Funktion `hallo()` gibt jetzt die englische Meldung aus. Wird umgekehrt auf den Verweis *deutsches Hallo-Welt* geklickt, so wird geprüft, ob die Datei *hallo\_deutsch.js* zugewiesen wurde und die Eigenschaft gegebenenfalls geändert. Die Ausgabe erfolgt dann in deutscher Sprache.

**Beachten Sie:** In Opera 5.12 und Netscape 6.1 ist dieses Beispiel nicht nachzuvollziehen, da beide Browser die neu zugewiesene Datei nicht interpretieren. Die Verwendung von Universaleigenschaften ist im HTML-4.0-Standard für das `script`-Element nicht vorgesehen. Deshalb sollten Sie bei diesem Element die Methode `document.getElementById()` vermeiden. Der MS Internet Explorer 5.0 Macintosh Edition reagiert mit einer Fehlermeldung bei der Verwendung dieses Beispiels, da er auf dieses Element offenbar nicht mit `document.getElementsByTagName()`

zugreifen kann. Opera 5.12 interpretiert das Attribut `type` nicht. Möchten Sie zum Beispiel einen Script-Bereich in *VB-Script* einfügen, so ist hier das missbilligte `language`-Attribut zur Sprachangabe zwingend erforderlich, da der Browser sonst Fehlermeldungen erzeugt, weil er als Default-Sprache JavaScript verwendet.



### *select (HTML-Elementobjekt)*

HTML-Elemente `<select>...</select>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft                | Status                        | Bedeutung                                                                                |
|----------------------------|-------------------------------|------------------------------------------------------------------------------------------|
| <code>disabled</code>      | <b>Lesen</b><br><b>Ändern</b> | Auswahlliste kann nicht benutzt werden                                                   |
| <code>form</code>          | <b>Lesen</b>                  | zugehöriges Formularelement                                                              |
| <code>length</code>        | <b>Lesen</b>                  | Anzahl der <code>option</code> -Elemente                                                 |
| <code>multiple</code>      | <b>Lesen</b><br><b>Ändern</b> | Mehrfachauswahl setzen oder nicht setzen                                                 |
| <code>name</code>          | <b>Lesen</b><br><b>Ändern</b> | Name der Auswahlliste                                                                    |
| <code>options</code>       | <b>Lesen</b>                  | Array der <code>option</code> -Elemente der Auswahlliste                                 |
| <code>selectedIndex</code> | <b>Lesen</b><br><b>Ändern</b> | Index des ausgewählten Eintrags                                                          |
| <code>size</code>          | <b>Lesen</b><br><b>Ändern</b> | Anzahl der sichtbaren Einträge                                                           |
| <code>tabIndex</code>      | <b>Lesen</b><br><b>Ändern</b> | Tabulatorenreihenfolge                                                                   |
| <code>type</code>          | <b>Lesen</b>                  | Typ der Auswahlliste: entweder <code>select-multiple</code> oder <code>select-one</code> |
| <code>value</code>         | <b>Lesen</b><br><b>Ändern</b> | Aktueller Auswahlwert                                                                    |

| Methode               | Bedeutung                            |
|-----------------------|--------------------------------------|
| <code>add()</code>    | Eintrag zur Auswahlliste hinzufügen  |
| <code>blur()</code>   | Fokus aus der Auswahlliste entfernen |
| <code>focus()</code>  | Fokus auf die Auswahlliste setzen    |
| <code>remove()</code> | Eintrag aus der Auswahlliste löschen |

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Wert = 1;
function hinzu() {
var position=null;
```

```

if (document.all)
position=position=document.getElementsByName("Auswahl")[0].length;
var Eintrag = document.createElement("option");
Eintrag.text = document.getElementsByName("Eingabe")[0].value;
Eintrag.value = Wert;
Wert += 1;

document.getElementsByName("Auswahl")[0].add(Eintrag,position);
}
//-->
</script>
</head><body>
<form name="Formular" action="">
<select name="Auswahl" size="10">
<option value="0">Liste</option>
</select><br>
Eintrag: <input name="Eingabe" size="30">
<input type="button" value="hinzufügen" onClick="hinzu()">
</form>
</body></html>

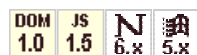
```

Das Beispiel enthält ein Formular mit einer Auswahlliste, einem Eingabefeld und einem Klick-Button. Klickt der Anwender auf den Button, wird der im Eingabefeld eingegebene Wert dynamisch der Auswahlliste als Eintrag hinzugefügt. Dazu wird die JavaScript-Funktion `hinzu()` aufgerufen, die im Dateikopf notiert ist. Bevor der Eintrag mit der Objektmethode `add()` hinzugefügt werden kann, muss zunächst mit `document.createElement()` ein neues `option`-Element erzeugt werden. Das Element wird in der Variablen `Eintrag` gespeichert. Seinen sichtbaren Text bekommt es mit `Eintrag.text` zugewiesen. Dabei wird mit `document.getElementsByName("Eingabe")[0].value` auf den Wert des Eingabefeldes zugegriffen. Auch einen Absendewert `value` erhält der Eintrag. Dazu wird die Variable `Wert` zugewiesen, die anschließend um 1 erhöht wird, sodass jeder neue Eintrag eine höhere Nummer als Absendewert erhält. Schließlich wird mit `document.getElementsByName("Auswahl")[0]` auf die Auswahlliste zugegriffen und der neue Eintrag am Ende der Liste hinzugefügt. Die Methode `add()` erwartet als ersten Parameter den Eintrag für das neue Listenelement und als zweiten Parameter die Position, vor der das Element eingefügt werden soll.

**Beachten Sie:** Innerhalb dieses Beispiels ist es erforderlich, zwischen den Browsern zu unterscheiden. Netscape 6.1 interpretiert die Methode `add()` nur dann, wenn ihr als Position `null` übergeben wird. Der Parameter `null` bedeutet übrigens: Eintrag am Ende einfügen.

Im MS Internet Explorer dagegen führt dieser Aufruf der Methode zu einem Typenkonflikt. Diesen können Sie vermeiden, indem Sie, wie im Beispiel, als Parameter die Länge der zu ergänzenden Liste übergeben oder alternativ den Parameter `position` weglassen.

Netscape 6.1 interpretiert die Angabe zur Position noch nicht wie vorgesehen. Um für beide Browser zu programmieren, müssen Sie also eine »Browser-Weiche« einfügen, z.B. mit Hilfe des `navigator`-Objekts.



### *small (HTML-Elementobjekt)*

HTML-Elemente `<small>...</small>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```

<html><head><title>Test</title>
<style type="text/css">
<!--
.mini { font-family:Small Fonts; font-size:5pt }
-->
</style>
<script type="text/javascript">
<!--
function ganzklein() {
  document.getElementsByTagName("small")[0].className = "mini";
}
//-->
</script>
</head><body>
<small onClick="ganzklein()">winzigweich!</small>
</body></html>

```

Das Beispiel enthält einen Text, der mit `<small>...</small>` ausgezeichnet ist. Beim Anklicken des Textes (`onClick`) wird die JavaScript-Funktion `ganzklein()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementsByTagName("small")[0]` auf das erste `small`-Element im Dokument zu und weist ihm die `className`-Eigenschaft mit dem Wert `mini` zu. Dahinter verbirgt sich die CSS-Klasse, die im Dateikopf im Style-Bereich definiert ist.



### *span (HTML-Elementobjekt)*

HTML-Elemente `<span>...</span>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

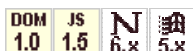
```

<html><head><title>Test</title>
<style type="text/css">
<!--
#diesesFormat { font-family:Arial; font-size:18pt }
#anderesFormat { font-family:Algerian; font-size:36pt }
-->
</style>
<script type="text/javascript">
<!--
function aendern() {
  document.getElementById("diesesFormat").id = "anderesFormat";
}
//-->
</script>
</head><body>
<span id="diesesFormat" onClick="aendern()">Das individuelle Format</span>
</body></html>

```

Das Beispiel enthält einen Text, der mit `<span>...</span>` ausgezeichnet ist. Das `span`-Element enthält außerdem eine CSS-Bindung, da es die `id` mit dem Wert `diesesFormat` hat, für das im Dateikopf ein individuelles Format definiert ist. Beim Anklicken des `span`-Textes

(onClick) wird die JavaScript-Funktion `aendern()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("diesesFormat")` auf das `span`-Element zu und weist ihm für die `id`-Eigenschaft den neuen Wert `anderesFormat` zu. Auch für diesen `id`-Wert ist im `Style`-Bereich eine `CSS`-Definition notiert. Das Beispiel bewirkt also ein einmaliges dynamisches Austauschen der `CSS`-Eigenschaften beim `span`-Element.



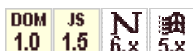
### ***strike*** (HTML-Elementobjekt)

HTML-Elemente `<strike>...</strike>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<strike id="strike" onMouseOver="alert(document.getElementById('strike').title)"
title="NICHT, weil es 'deprecated' ist">Benutzen Sie dieses Element!</strike>
</body></html>
```

Das Beispiel enthält einen Text, der mit `<strike>...</strike>` ausgezeichnet ist. Im einleitenden Tag ist der Event-Handler `onMouseOver` notiert. Beim Überfahren des durchgestrichenen Textes mit der Maus wird eine Meldung ausgegeben, und zwar der Text, der beim Element im `title`-Attribut notiert ist. Dazu wird mit `document.getElementById('strike').title` auf das Element zugegriffen. Der `id`-Name `strike` lautet nur zufällig genauso wie der Elementname, um zu demonstrieren, dass dies durchaus erlaubt ist.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



### ***strong*** (HTML-Elementobjekt)

HTML-Elemente `<strong>...</strong>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
.normal { font-size:20pt; color:blue; }
.extra { font-size:20pt; color:red; background-color:yellow; }
-->
</style>
<script type="text/javascript">
<!--
function wechseln() {
  if(document.getElementById("stark").className == "normal")
    document.getElementById("stark").className = "extra";
  else
    document.getElementById("stark").className = "normal";
}
//-->
</script>
</head><body>
<p><strong id="stark" class="normal" onMouseOver="wechseln()"
onMouseOut="wechseln()">ganz stark!</strong></p>
</body></html>
```

Das Beispiel enthält Text, der mit `<strong>...</strong>` ausgezeichnet ist. Das einleitende Tag enthält eine CSS-Klassenangabe, eine eindeutige Id-Angabe und die Event-Handler `onMouseOver` und `onMouseOut`, die jeweils die JavaScript-Funktion `wechseln()` aufrufen, die im Dateikopf notiert ist. Diese Funktion fragt ab, ob der zugewiesene Klassenname `normal` lautet. Wenn ja, wird er auf `extra` geändert, wenn nein, wird er auf `normal` geändert. Dadurch ergibt sich der Effekt, dass sich beim Überfahren des Textes mit der Maus die CSS-Eigenschaften dynamisch ändern.



### style (HTML-Elementobjekt)

HTML-Elemente `<style>...</style>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                          | Bedeutung                                 |
|-------------|-------------------------------------------------|-------------------------------------------|
| disabled    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Stylesheet verwenden oder nicht verwenden |
| media       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausgabemedium für das Stylesheet          |
| type        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Mime-Type der Stylesheet-Sprache          |

Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
h1 { font-size:36pt; color:red; }
-->
</style>
<script type="text/javascript">
<!--
function ohneCSS() {
    document.getElementsByTagName("style")[0].disabled = true;
}
//-->
</script>
</head><body>
<h1>Mit CSS!</h1>
<a href="javascript:ohneCSS()"><b>ohne CSS!</b></a>
</body></html>
```

Das Beispiel enthält eine Überschrift und einen Verweis. Beim Anklicken des Verweises wird die JavaScript-Funktion `ohneCSS()` aufgerufen, die im Dateikopf notiert ist. Die Funktion greift mit `document.getElementsByTagName("style")[0]` auf das erste `style`-Element zu und setzt dessen Eigenschaft `disabled` auf den booleschen Wert `true` (»wahr«). Dadurch werden alle in dem Style-Bereich definierten Formate für das Dokument ungültig. Die Überschrift verliert dynamisch ihre CSS-Eigenschaften.



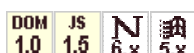
**sub (HTML-Elementobjekt)**

HTML-Elemente `<sub>...</sub>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:alert(document.getElementsByTagName('sub')[0].title)">Tief</a>
und <sub title="ganz tief">tiefer</sub>
</body></html>
```

Das Beispiel enthält ein Wort, das mit `<sub>...</sub>` ausgezeichnet ist, und einen Verweis. Beim Anklicken des Verweises wird in einem Meldungsfenster der Wert des `title`-Attributs aus dem `sub`-Element ausgegeben. Dazu wird mit `document.getElementsByTagName('sub')[0]` auf das erste `sub`-Element der Datei zugegriffen.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.

**sup (HTML-Elementobjekt)**

HTML-Elemente `<sup>...</sup>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Vertauschen() {
  var swap = document.getElementById("Basis").firstChild.nodeValue;
  document.getElementById("Basis").firstChild.nodeValue =
    document.getElementById("Exponent").firstChild.nodeValue;
  document.getElementById("Exponent").firstChild.nodeValue = swap;
}
//-->
</script>
</head><body><span style="font-size:200%">
<a id="Basis" href="javascript:Vertauschen()">x</a><sup id="Exponent">y</sup>
</span></body></html>
```

Das Beispiel stellt einen Text »x hoch y« dar. Das »x« ist dabei in einen Verweis eingeschlossen, bei dessen Anklicken die JavaScript-Funktion `Vertauschen()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion greift sowohl auf das Verweis-Element (`document.getElementById("Basis")`) als auch auf das `sup`-Element (`document.getElementById("Exponent")`) zu und vertauscht deren Inhalte. Der Zugriff auf die Inhalte erfolgt mit `node.nodeValue` (`firstChild` ist ein bestimmter Knotentyp).

**table (HTML-Elementobjekt)**

HTML-Elemente `<table>...</table>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft | Status                        | Bedeutung                                                                  |
|-------------|-------------------------------|----------------------------------------------------------------------------|
| align       | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung der Tabelle                                                    |
| border      | <b>Lesen</b><br><b>Ändern</b> | Rahmendicke der Tabelle                                                    |
| bgColor     | <b>Lesen</b><br><b>Ändern</b> | Hintergrundfarbe der Tabelle                                               |
| caption     | <b>Lesen</b><br><b>Ändern</b> | Tabellenüberschrift-Element vorhanden oder nicht vorhanden                 |
| cellPadding | <b>Lesen</b><br><b>Ändern</b> | Abstand zwischen Zellenrand und Zellinhalt                                 |
| cellSpacing | <b>Lesen</b><br><b>Ändern</b> | Abstand zwischen den Zellen untereinander                                  |
| frame       | <b>Lesen</b><br><b>Ändern</b> | Angabe, welche Rahmen der Tabelle sichtbar sein sollen                     |
| rows        | <b>Lesen</b>                  | Array aller Tabellenzeilen                                                 |
| rules       | <b>Lesen</b><br><b>Ändern</b> | Angabe, welche Gitternetzlinien innerhalb der Tabelle sichtbar sein sollen |
| summary     | <b>Lesen</b><br><b>Ändern</b> | Text, der Sehbehinderten die Tabelle im Fall von Sprachausgabe erläutert   |
| tBodies     | <b>Lesen</b>                  | Array aller tbody-Elemente der Tabelle                                     |
| tFoot       | <b>Lesen</b><br><b>Ändern</b> | Tabellenfuß-Element oder null                                              |
| tHead       | <b>Lesen</b><br><b>Ändern</b> | Tabellenkopf-Element oder null                                             |
| width       | <b>Lesen</b><br><b>Ändern</b> | Breite der Tabelle                                                         |

| Methode         | Bedeutung                                                 |
|-----------------|-----------------------------------------------------------|
| createCaption() | Tabellenüberschrift-Element <code>caption</code> erzeugen |
| createTFoot()   | Tabellenfuß-Element <code>tfoot</code> erzeugen           |
| createTHead()   | Tabellenkopf-Element <code>thead</code> erzeugen          |
| deleteCaption() | Tabellenüberschrift-Element <code>caption</code> löschen  |
| deleteTFoot()   | Tabellenfuß-Element <code>tfoot</code> löschen            |
| deleteTHead()   | Tabellenkopf-Element <code>thead</code> löschen           |
| deleteRow()     | Tabellenzeile löschen                                     |
| insertRow()     | Tabellenzeile hinzufügen                                  |

### Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zeile = 0;
function ZeileEinfuegen() {
  if (Zeile==0) document.getElementById("Tabelle").deleteRow(0);
  var TR = document.getElementById("Tabelle").insertRow(Zeile);

```

```

Zeile += 1;
var TD1 = document.createElement("td");
var TD1text = document.createTextNode(document.Formular.linkerText.value);
TD1.appendChild(TD1text);
var TD2 = document.createElement("td");
var TD2text = document.createTextNode(document.Formular.rechterText.value);
TD2.appendChild(TD2text);
TR.appendChild(TD1);
TR.appendChild(TD2);
document.Formular.linkerText.value = "";
document.Formular.rechterText.value = "";
}
function Rahmen() {
  if(document.getElementsByName("Option")[0].checked == true)
    document.getElementById("Tabelle").border = "2";
  else
    document.getElementById("Tabelle").border = "0";
}
function Farbe() {
  if(document.getElementsByName("Option")[1].checked == true)
    document.getElementById("Tabelle").bgColor = "#CCCCFF";
  else
    document.getElementById("Tabelle").bgColor = "";
}
//-->
</script>
</head><body>
<form name="Formular" action="">
<input type="text" name="linkerText">
<input type="text" name="rechterText">
<input type="button" value=" OK " onClick="ZeileEinfuegen()"><br>
<input type="checkbox" name="Option" onClick="Rahmen()"> Rahmen
<input type="checkbox" name="Option" onClick="Farbe()"> Hintergrund
</form>
<table id="Tabelle">
<tr><td>Eine Tabelle</td></tr>
</table>
</body></html>

```

Das Beispiel enthält ein Formular und unterhalb davon eine einzeilige Tabelle. Das Formular besteht aus zwei Eingabefeldern, einem Klick-Button und zwei Checkboxes. Die beiden Eingabefelder dienen dazu, um die Tabelle mit neuem Inhalt zu füllen. Gibt der Anwender in beiden Feldern etwas ein und klickt dann auf den Klick-Button, wird die JavaScript-Funktion `ZeileEinfuegen()` aufgerufen, die im Dateikopf notiert ist. Beim ersten Aufruf der Funktion – die Variable `Zeile` ist hier noch 0 – wird mit `document.getElementById("Tabelle").deleteRow(0)` die vorhandene Tabellenzeile gelöscht.

Mit `document.getElementById("Tabelle").insertRow(Zeile)` greift die Funktion auf die Tabelle zu und fügt ihr mit `insertRow()` eine neue Zeile ein. Der übergebene Wert `Zeile` bestimmt, an welcher Stelle der Tabelle die neue Zeile eingefügt werden soll. Da die Variable

Zeile anschließend um 1 erhöht wird, wird beim nächsten Aufruf der Funktion die Zeile unterhalb der letzten Zeile hinzugefügt.

Der Rückgabewert von `insertRow()` ist das neu erzeugte `tr`-Element für die Tabellenzeile. Das Element wird in der Variablen `TR` gespeichert. Um die Tabellenzeile mit Inhalt zu füllen, werden zwei Tabellenzellen erzeugt. Die Elemente werden mit `document.createElement()` erzeugt. Für die Inhalte der Tabellenzellen werden mit `document.createTextNode()` zwei Textknoten erzeugt, die als Wert die Inhalte der beiden Eingabefelder des Formulars zugewiesen bekommen.

Mit der Methode `node.appendChild()` werden die Textknoten als Kindknoten zu den Tabellenzellen hinzugefügt, und anschließend wird mit der gleichen Methode die Tabellenzelle als Kindknoten zu der Tabellenzeile hinzugefügt.

Die beiden Checkboxes rufen beim Anklicken (`onClick`) die Funktionen `Rahmen()` bzw. `Farbe()` auf. Dort wird jeweils überprüft, ob die Checkboxes angekreuzt sind (`checked`) oder nicht. Dann wird mit `document.getElementById("Tabelle")` auf die Tabelle zugegriffen und abhängig davon, ob die Checkbox angekreuzt ist oder nicht, werden deren Eigenschaften für `border` bzw. `bgColor` dynamisch geändert.



### **tbody (HTML-Elementobjekt)**

HTML-Elemente `<tbody>...</tbody>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft         | Status                        | Bedeutung                                                        |
|---------------------|-------------------------------|------------------------------------------------------------------|
| <code>align</code>  | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung der Zellen des Tabellenbereichs                      |
| <code>ch</code>     | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen für Zellen des Tabellenbereichs              |
| <code>chOff</code>  | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens in Zellen des Tabellenbereichs |
| <code>rows</code>   | <b>Lesen</b>                  | Array aller Tabellenzeilen des Tabellenbereichs                  |
| <code>vAlign</code> | <b>Lesen</b><br><b>Ändern</b> | vertikale Ausrichtung der Zellen im Tabellenbereich              |

| Methode                  | Bedeutung                                   |
|--------------------------|---------------------------------------------|
| <code>deleteRow()</code> | Tabellenzeile aus Tabellenbereich löschen   |
| <code>insertRow()</code> | Tabellenzeile im Tabellenbereich hinzufügen |

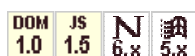
Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function rechts() {
    document.getElementById("Tabellendaten").align = "right";
}
```

```
//-->
</script>
</head><body>
<table border="1">
<tr><th width="200">Name</th><th width="200">Alter</th></tr>
<tbody id="Tabellendaten">
<tr><td>Hans</td><td>18</td></tr>
<tr><td>Flora</td><td>17</td></tr>
</tbody>
</table>
<a href="javascript:rechts()">rechts ausrichten</a>
</body></html>
```

Das Beispiel enthält eine Tabelle mit ausgewiesenem tbody-Bereich. Unterhalb der Tabelle ist ein Verweis notiert, bei dessen Anklicken die JavaScript-Funktion `rechts()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("Tabellendaten")` auf das tbody-Element zu und ändert dessen Eigenschaft `align` auf den Wert `right`. Dadurch werden alle Tabellenzellen im tbody-Bereich rechtsbündig ausgerichtet.

**Beachten Sie:** Mit Netscape 6.1 und dem MS Internet Explorer 5.0 Macintosh Edition war dieses Beispiel nicht nachvollziehbar.



### td (HTML-Elementobjekt)

HTML-Elemente `<td>...</td>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

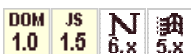
| Eigenschaft | Status                        | Bedeutung                                                                                                             |
|-------------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| abbr        | <b>Lesen</b><br><b>Ändern</b> | Hinweistext für die Zuordnung der Tabellenzelle, die z.B. bei der Sprachausgabe dem Zelleninhalt vorangesprochen wird |
| align       | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung                                                                                                           |
| axis        | <b>Lesen</b><br><b>Ändern</b> | Liste von Kategorien, zu der die Zelle gehört                                                                         |
| bgColor     | <b>Lesen</b><br><b>Ändern</b> | Hintergrundfarbe                                                                                                      |
| cellIndex   | <b>Lesen</b>                  | Indexnummer der Zelle innerhalb der Tabellenzeile (Start bei 0)                                                       |
| ch          | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen                                                                                                   |
| chOff       | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens                                                                                     |
| colSpan     | <b>Lesen</b><br><b>Ändern</b> | Anzahl Spalten, über die sich die Zelle erstreckt                                                                     |
| headers     | <b>Lesen</b><br><b>Ändern</b> | Liste von ID-Namen von Zeilen- oder Spaltenüberschriften, zu denen die Zelle gehört                                   |
| height      | <b>Lesen</b><br><b>Ändern</b> | Zellenhöhe (Zeilenhöhe)                                                                                               |

| Eigenschaft | Status                                          | Bedeutung                                                                        |
|-------------|-------------------------------------------------|----------------------------------------------------------------------------------|
| noWrap      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Automatischer Zeilenumbruch oder kein automatischer Zeilenumbruch                |
| rowSpan     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzahl Tabellenzeilen, über die sich die Zelle erstreckt                         |
| scope       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | restliche Zeilen oder Spalten, für die aktuelle Zelle Überschriftencharakter hat |
| vAlign      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | vertikale Ausrichtung                                                            |
| width       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Zellenbreite (Spaltenbreite)                                                     |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<table cellspacing="2" border="2">
<tr><td>Hans</td><td>18</td></tr>
<tr><td>Flora</td><td>17</td></tr>
</table>
<script type="text/javascript">
<!--
var Farben = new Array("#CCFFCC", "#CCCCFF", "FFCCCC", "FFCCFF", "FFFFCC");
for(var i = 0; i < document.getElementsByTagName("td").length; i++) {
  var index = parseInt((Math.random() * 100) % (Farben.length - 1));
  document.getElementsByTagName("td")[i].bgColor = Farben[index];
}
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Tabelle. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird in einer for-Schleife der Reihe nach mit `document.getElementsByTagName("td")` auf alle td-Elemente der Datei zugegriffen. Durch Aufruf von `Math.random()` werden Zufallszahlen ermittelt, die mit Hilfe von Multiplikation mit 100, Modulodivision und Anwendung der Funktion `parseInt()` so getrimmt werden, dass es ganzzahlige Zufallszahlen werden, die sich als Index für den zuvor definierten Array `Farben` eignen. Mit `document.getElementsByTagName("td")[i].bgColor` wird schließlich die Hintergrundfarbe der jeweils aktuellen Tabellenzelle auf einen zufällig ermittelten Farbwert gesetzt. Die Tabelle wird auf diese Weise schön bunt, und jede Zelle erhält eine per Zufall ausgesuchte Hintergrundfarbe.



### textarea (HTML-Elementobjekt)

HTML-Elemente `<textarea>...</textarea>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft  | Status                                          | Bedeutung                                     |
|--------------|-------------------------------------------------|-----------------------------------------------|
| accessKey    | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Hotkey für das Element                        |
| cols         | <a href="#">Lesen</a>                           | Zeichen pro Zeile (Anzeigebreite)             |
| defaultValue | <a href="#">Lesen</a><br><a href="#">Ändern</a> | vorbelegter Wert                              |
| disabled     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Element kann nicht geändert werden            |
| form         | <a href="#">Lesen</a>                           | zugehöriges Formular                          |
| name         | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Name des Elements                             |
| readOnly     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Wert des Elements kann nicht verändert werden |
| rows         | <a href="#">Lesen</a>                           | Zeilen (Anzeighöhe)                           |
| tabIndex     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Tabulatorreihenfolge                          |
| type         | <a href="#">Lesen</a>                           | Typ des Formularelements                      |
| value        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Wert des Elements                             |

| Methode  | Bedeutung                   |
|----------|-----------------------------|
| blur()   | Fokus vom Element entfernen |
| focus()  | Fokus auf Element setzen    |
| select() | Inhalt selektieren          |

#### Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function setText() {
    document.getElementById("Text").firstChild.nodeValue =
document.getElementById("Eingabe").value;
}
//-->
</script>
</head><body>
<form name="Formular" action="">
Text:<br>
<textarea id="Eingabe" rows="10" cols="50"
onKeyPress="setText()"></textarea>
</form>
<div id="Text" style="font-size:16pt; background-color:#FFFFCC; padding:10px;
border:solid 1px red">&nbsp;&nbsp;&nbsp;</div>
</body></html>

```

Das Beispiel enthält ein Formular mit einem `textarea`-Eingabebereich. Unterhalb des Formulars ist ein schick formatierter `div`-Bereich notiert, der als Inhalt aber zunächst nur ein erzwungenes Leerzeichen `&nbsp;` enthält. Im einleitenden `<textarea>`-Tag ist der Event-Handler `onKeyPress` notiert. Wenn der Anwender in dem Feld etwas eingibt, wird bei jedem Loslassen eines Tastendrucks die JavaScript-Funktion `setText()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("Text").firstChild.nodeValue` auf den Inhalt des `div`-Elements zu und weist diesem den Wert des Eingabefeldes (`document.getElementById("Eingabe").value`) zu. Somit wird in den `div`-Bereich dynamisch der Text geschrieben, den der Anwender im Textfeld eingibt.



### **tfoot (HTML-Elementobjekt)**

HTML-Elemente `<tfoot>...</tfoot>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft         | Status                        | Bedeutung                                                        |
|---------------------|-------------------------------|------------------------------------------------------------------|
| <code>align</code>  | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung der Zellen des Tabellenbereichs                      |
| <code>ch</code>     | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen für Zellen des Tabellenbereichs              |
| <code>chOff</code>  | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens in Zellen des Tabellenbereichs |
| <code>rows</code>   | <b>Lesen</b>                  | Array aller Tabellenzeilen des Tabellenbereichs                  |
| <code>vAlign</code> | <b>Lesen</b><br><b>Ändern</b> | vertikale Ausrichtung der Zellen im Tabellenbereich              |

| Methode                  | Bedeutung                                   |
|--------------------------|---------------------------------------------|
| <code>deleteRow()</code> | Tabellenzeile aus Tabellenbereich löschen   |
| <code>insertRow()</code> | Tabellenzeile im Tabellenbereich hinzufügen |

### Ein Beispiel:

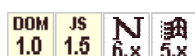
```
<html><head><title>Test</title>
</head><body>
<table border="1">
<tfoot id="Fuss" valign="top">
<tr><td colspan="2" height="40">
Angaben ohne Gewähr!
</td></tr></tfoot>
<tr><td>Hans</td><td>18</td></tr>
<tr><td>Flora</td><td>17</td></tr>
</table>
<script type="text/javascript">
<!--
var Ausrichtung = document.getElementById("Fuss").vAlign;
document.write("Angaben ohne Gewähr haben die Ausrichtung: " + Ausrichtung);
```



```
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Tabelle mit einem ausgewiesenen tfoot-Bereich. Unterhalb der Tabelle ist ein JavaScript-Bereich notiert. Darin wird mit `document.getElementById("Fuss").vAlign` auf das tfoot-Element zugegriffen und der Wert von dessen Eigenschaft `vAlign` ausgelesen. Anschließend wird das Ergebnis ins Dokument geschrieben.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



### th (HTML-Elementobjekt)

HTML-Elemente `<th>...</th>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

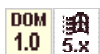
| Eigenschaft | Status                                          | Bedeutung                                                                                                             |
|-------------|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| abbr        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Hinweistext für die Zuordnung der Tabellenzelle, die z.B. bei der Sprachausgabe dem Zelleninhalt vorangesprochen wird |
| align       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausrichtung                                                                                                           |
| axis        | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Liste von Kategorien, zu der die Zelle gehört                                                                         |
| bgColor     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Hintergrundfarbe                                                                                                      |
| cellIndex   | <a href="#">Lesen</a>                           | Indexnummer der Zelle innerhalb der Tabellenzeile (Start bei 0)                                                       |
| ch          | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Ausrichtungszeichen                                                                                                   |
| chOff       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Position des Ausrichtungszeichens                                                                                     |
| colSpan     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzahl Spalten, über die sich die Zelle erstreckt                                                                     |
| headers     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Liste von ID-Namen von Zeilen- oder Spaltenüberschriften, zu denen die Zelle gehört                                   |
| height      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Zellenhöhe (Zeilenhöhe)                                                                                               |
| noWrap      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | automatischer Zeilenumbruch oder kein automatischer Zeilenumbruch                                                     |
| rowSpan     | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Anzahl Tabellenzeilen, über die sich die Zelle erstreckt                                                              |
| scope       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | restliche Zeilen oder Spalten, für die aktuelle Zelle Überschriftencharakter hat                                      |
| vAlign      | <a href="#">Lesen</a><br><a href="#">Ändern</a> | vertikale Ausrichtung                                                                                                 |
| width       | <a href="#">Lesen</a><br><a href="#">Ändern</a> | Zellenbreite (Spaltenbreite)                                                                                          |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<table border="1">
<tr><th width="200">Name</th><th width="200">Alter</th></tr>
<tr><td>Hans</td><td>18</td></tr>
<tr><td>Flora</td><td>17</td></tr>
</table>
<script type="text/javascript">
<!--
for(var i = 0; i < document.getElementsByTagName("th").length; i++)
  document.getElementsByTagName("th")[i].align = "left";
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Tabelle und unterhalb davon einen JavaScript-Bereich. Dort werden in einer `for`-Schleife der Reihe nach alle `th`-Elemente des Dokuments ermittelt. Ihnen wird für die Eigenschaft `align` der Wert `left` zugewiesen.

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.



### *thead* (HTML-Elementobjekt)

HTML-Elemente `<thead>...</thead>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft         | Status                        | Bedeutung                                                        |
|---------------------|-------------------------------|------------------------------------------------------------------|
| <code>align</code>  | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung der Zellen des Tabellenbereichs                      |
| <code>ch</code>     | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen für Zellen des Tabellenbereichs              |
| <code>chOff</code>  | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens in Zellen des Tabellenbereichs |
| <code>rows</code>   | <b>Lesen</b>                  | Array aller Tabellenzeilen des Tabellenbereichs                  |
| <code>vAlign</code> | <b>Lesen</b><br><b>Ändern</b> | vertikale Ausrichtung der Zellen im Tabellenbereich              |

| Methode                  | Bedeutung                                   |
|--------------------------|---------------------------------------------|
| <code>deleteRow()</code> | Tabellenzeile aus Tabellenbereich löschen   |
| <code>insertRow()</code> | Tabellenzeile im Tabellenbereich hinzufügen |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<table border="1">
```

```

<thead id="Kopf">
<tr><th width="200">Name</th><th width="200">Alter</th></tr>
</thead>
<tr><td>Hans</td><td>18</td></tr>
<tr><td>Flora</td><td>17</td></tr>
</table>
<script type="text/javascript">
<!--
  var Zeilen = document.getElementById("Kopf").rows.length;
  document.write("Tabellenkopf hat " + Zeilen + " Zeilen");
//-->
</script>
</body></html>

```

Das Beispiel enthält eine Tabelle und unterhalb davon einen JavaScript-Bereich. Dort wird mit `document.getElementById("Kopf").rows.length` die Anzahl der Tabellenzeilen im Bereich des `thead`-Elements ermittelt und anschließend ins Dokument geschrieben.



### *title (HTML-Elementobjekt)*

HTML-Elemente `<title>...</title>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgende eigenen Eigenschaft.

| Eigenschaft | Status                                                                                                                                                                                                | Bedeutung         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| text        |  <b>Lesen</b><br> <b>Ändern</b> | Inhalt des Titels |

Ein Beispiel:

```

<html><head><title>Ein ganz besonderer Tag</title>
</head><body>
<script type="text/javascript">
<!--
  document.write("<h1>" + document.getElementsByTagName('title')[0].text +
"</h1>");
//-->
</script>
</body></html>

```

Das Beispiel enthält einen JavaScript-Bereich, in dem dynamisch eine Überschrift erster Ordnung ins Dokument geschrieben wird. Als Überschriftentext wird der Inhalt des `title`-Elements geschrieben. Dazu wird mit `document.getElementsByTagName('title')[0]` auf das `title`-Element zugegriffen.



### *tr (HTML-Elementobjekt)*

HTML-Elemente `<tr>...</tr>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften und Methoden.

| Eigenschaft     | Status                        | Bedeutung                                                                                            |
|-----------------|-------------------------------|------------------------------------------------------------------------------------------------------|
| align           | <b>Lesen</b><br><b>Ändern</b> | Ausrichtung der Zellen der Tabellenzeile                                                             |
| bgColor         | <b>Lesen</b><br><b>Ändern</b> | Hintergrundfarbe der Zellen der Tabellenzeile                                                        |
| cells           | <b>Lesen</b>                  | Array der Tabellenzellen dieser Tabellenzeile                                                        |
| ch              | <b>Lesen</b><br><b>Ändern</b> | Ausrichtungszeichen für Zellen der Tabellenzeile                                                     |
| chOff           | <b>Lesen</b><br><b>Ändern</b> | Position des Ausrichtungszeichens in Zellen der Tabellenzeile                                        |
| rowIndex        | <b>Lesen</b>                  | Indexnummer der Tabellenzeile (startend bei 0) relativ zur Tabelle                                   |
| sectionRowIndex | <b>Lesen</b>                  | Indexnummer der Tabellenzeile (startend bei 0) relativ zum Tabellenbereich (thead, tbody oder tfoot) |
| vAlign          | <b>Lesen</b><br><b>Ändern</b> | vertikale Ausrichtung der Zellen im Tabellenbereich                                                  |

| Methode      | Bedeutung                                 |
|--------------|-------------------------------------------|
| deleteCell() | Tabellenzelle aus Tabellenzeile löschen   |
| insertCell() | Tabellenzelle in Tabellenzeile hinzufügen |

### Ein Beispiel:

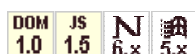
```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zelle = 0;
function neueZelle() {
  if (Zelle==0) document.getElementById("Zeile").deleteCell(0);
  var Text = document.createTextNode(document.Formular.Eingabe.value);
  var TD = document.getElementById("Zeile").insertCell(Zelle);
  Zelle += 1;
  TD.appendChild(Text);
  document.Formular.Eingabe.value = "";
}
//-->
</script>
</head><body>
<table border="1">
<tr id="Zeile"><td>Zelle</td></tr>
</table>
<form name="Formular" action="">
<input type="text" name="Eingabe">
<input type="button" value=" OK " onClick="neueZelle()">
</form>
</body></html>

```

Das Beispiel enthält eine Tabelle mit einer einspaltigen Tabellenzeile und unterhalb der Tabelle ein Formular mit einem Eingabefeld und einem Klick-Button. Beim Anklicken des Buttons wird die JavaScript-Funktion `neueZeile()` aufgerufen, die im Dateikopf notiert ist. Beim ersten Aufruf der Funktion – die Variable `Zeile` ist hier noch 0 – wird mit `document.getElementById("Zeile").deleteCell(0)` die vorhandene Tabellenzelle gelöscht. Die Funktion erzeugt zunächst mit `document.createTextNode()` einen Textknoten, der als Wert den Inhalt des Formular-Eingabefeldes erhält. Anschließend wird mit `document.getElementById("Zeile")` auf die Tabellenzeile zugegriffen und mit der Methode `insertCell()` eine neue Tabellenzelle eingefügt. Der übergebene Parameter bestimmt, an wievielter Stelle die Zelle in der Zeile eingefügt wird. Im Beispiel wird dazu die Variable `Zeile` benutzt, die bei jedem Funktionsaufruf um 1 erhöht wird. So ist sichergestellt, dass neue Zellen immer ans Ende der Zeile angehängt werden.

Der Rückgabewert von `insertCell()` ist das neu erzeugte `td`-Element. Es wird in der Variablen `TD` gespeichert. Mit der Methode `node.appendChild()` wird der zuvor erzeugte Textknoten in das `td`-Element als Kindknoten hinzugefügt. Die Tabellenzeile ist damit dynamisch um das neue Zellelement erweitert worden.



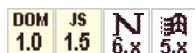
#### *tt* (HTML-Elementobjekt)

HTML-Elemente `<tt>...</tt>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Unversaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<tt title="ein Teletyper!"
onmouseover="alert(document.getElementsByTagName('tt')[0].title)">Was bin ich?</tt>
</body></html>
```

Das Beispiel enthält mit `<tt>...</tt>` ausgezeichneten Text, der den Event-Handler `onmouseover` enthält. Beim Überfahren des Textes mit der Maus wird in einem Meldungsfenster der Wert des `title`-Attributs des Elements ausgegeben. Dazu wird mit `document.getElementsByTagName('tt')[0]` auf das Element zugegriffen.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.



#### *u* (HTML-Elementobjekt)

HTML-Elemente `<u>...</u>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Unversaleigenschaften. Ein Beispiel:

```
<html><head><title>Ein ganz besonderer Tag</title>
<style type="text/css">
<!--
.so { text-decoration:overline; }
-->
</style>
<script type="text/javascript">
<!--
function anders() {
document.getElementById("uText").className = "so";
```

```

}
//-->
</script>
</head><body>
<u id="uText" onClick="anders()">das ist der Text</u>
</body></html>

```

Das Beispiel enthält unterstrichenen Text. Im einleitenden `<u>`-Tag ist der Event-Handler `onClick` notiert. Beim Anklicken des Textes dieses Elements wird die JavaScript-Funktion `anders()` aufgerufen. Diese greift mit `document.getElementById("uText")` auf das `u`-Element zu und weist ihm die im Dateikopf in einem Style-Bereich definierte Klasse `so` zu. Der Text wird dadurch überstrichen statt unterstrichen dargestellt.

**Beachten Sie:** Im MS Internet Explorer 5.0 wird als Ergebnis des Funktionsaufrufes der Text sowohl unterstrichen als auch überstrichen dargestellt. Netscape 6.1 und der MS Internet Explorer 5.5 verhalten sich dagegen wie im Beispiel beschrieben.

#### **ul (HTML-Elementobjekt)**

HTML-Elemente `<ul>...</ul>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften sowie die folgenden eigenen Eigenschaften.

| Eigenschaft | Status                                                                              | Bedeutung                    |
|-------------|-------------------------------------------------------------------------------------|------------------------------|
| compact     |  | Darstellung in enger Schrift |
| type        |  | Art des Aufzählungszeichens  |

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Bullet(Symbol) {
  document.getElementById("Liste").type = Symbol;
}
//-->
</script>
</head><body>
<ul id="Liste">
<li>ein Punkt</li>
<li>und noch ein Punkt</li>
</ul>
<a href="javascript:Bullet('disc')">Disc-Symbol</a>
<a href="javascript:Bullet('square')">Square-Symbol</a>
</body></html>

```

Das Beispiel enthält eine Aufzählungsliste. Unterhalb davon sind zwei Verweise notiert. Bei Anklicken der Verweise wird jeweils die JavaScript-Funktion `Bullet()` aufgerufen, die im Datei-

kopf notiert ist. Diese Funktion greift mit `document.getElementById("Liste")` auf das `ul`-Element der Liste zu und weist seiner Eigenschaft `type` den Wert zu, den die Funktion beim Aufruf von den Verweisen übergeben bekommen hat.



### **var (HTML-Elementobjekt)**

HTML-Elemente `<var>...</var>` haben als DOM-Objekte für den Scriptsprachen-Zugriff Universaleigenschaften. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
Datei: <var id="datei"
title="/usr/docs/web/projekte/dateiname.html">dateiname.html</var><br>
genauer:
<script type="text/javascript">
<!--
  document.write("<var>" + document.getElementById('datei').title + "<\var>");
-->
</script>
</body></html>
```

Das Beispiel enthält einen mit `<var>...</var>` ausgezeichneten Text. Unterhalb ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementById('datei')` auf das `var`-Element zugegriffen und der Wert von dessen Eigenschaft `title` dynamisch ins Dokument geschrieben.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.

## 20.6 node

### **Eigenschaften**

|                              |                                |
|------------------------------|--------------------------------|
| <code>attributes</code>      | (Attribute)                    |
| <code>childNodes</code>      | (Kindknoten)                   |
| <code>data</code>            | (Zeichendaten)                 |
| <code>firstChild</code>      | (erster Kindknoten)            |
| <code>lastChild</code>       | (letzter Kindknoten)           |
| <code>nextSibling</code>     | (nächster Knoten eines Typs)   |
| <code>nodeName</code>        | (Name des Knotens)             |
| <code>nodeType</code>        | (Knotentyp)                    |
| <code>nodeValue</code>       | (Wert/Inhalt des Knotens)      |
| <code>parentNode</code>      | (Elternknoten)                 |
| <code>previousSibling</code> | (vorheriger Knoten eines Typs) |

### **Methoden**

|                            |                           |
|----------------------------|---------------------------|
| <code>appendChild()</code> | (Kindknoten hinzufügen)   |
| <code>appendData()</code>  | (Zeichendaten hinzufügen) |
| <code>cloneNode()</code>   | (Knoten kopieren)         |
| <code>deleteData()</code>  | (Zeichendaten löschen)    |

|                                    |                                        |
|------------------------------------|----------------------------------------|
| <code>getAttribute()</code>        | (Wert eines Attributknotens ermitteln) |
| <code>getAttributeNode()</code>    | (Attributknoten ermitteln)             |
| <code>hasChildNodes()</code>       | (auf Kindknoten prüfen)                |
| <code>insertBefore()</code>        | (Knoten einfügen)                      |
| <code>insertData()</code>          | (Zeichendaten einfügen)                |
| <code>removeAttribute()</code>     | (Wert eines Attributknotens löschen)   |
| <code>removeAttributeNode()</code> | (Attributknoten löschen)               |
| <code>removeChild()</code>         | (Knoten löschen)                       |
| <code>replaceChild()</code>        | (Kindknoten ersetzen)                  |
| <code>replaceData()</code>         | (Zeichendaten ersetzen)                |
| <code>setAttribute()</code>        | (Wert eines Attributknotens setzen)    |
| <code>setAttributeNode()</code>    | (Attributknoten erzeugen)              |

### ***node: Allgemeines zur Verwendung***

Das `node`-Objekt ist das zentrale Objekt des Document Object Model (DOM) (*node* = *Knoten*). Hintergrund ist das Modell, dass ein Auszeichnungssprachen-Dokument, egal ob in HTML oder einer anderen XML-basierten Auszeichnungssprache geschrieben, aus einem Baum von Knoten besteht.

Jedes Element, jedes Attribut und alle Zeichendaten stellen eigene Knoten dar. Diese Knoten bilden die Baumstruktur. Das `node`-Objekt stellt Eigenschaften und Methoden bereit, um auf die einzelnen Knoten zuzugreifen, egal, wie tief diese Knoten in der Baumstruktur liegen.

Das `node`-Objekt stellt damit die allgemeinere und für alle XML-gerechten Sprachen gültige Variante dessen dar, was die HTML-Elementobjekte speziell für HTML darstellen. Sie können in JavaScripts, die in HTML-Dateien notiert oder eingebunden sind, sowohl mit den HTML-Elementobjekten als auch mit dem `node`-Objekt arbeiten. Manches ist über die HTML-Elementobjekte bequemer zu lösen, für andere Aufgaben eignet sich wiederum das `node`-Objekt besser. Das `node`-Objekt gilt unter Puristen allerdings als das »reinere« DOM, eben weil es nicht auf HTML beschränkt ist.

Um auf die Eigenschaften und Methoden des `node`-Objekts zugreifen zu können, benötigen Sie einen Knoten. Um auf vorhandene Knoten im Dokument zuzugreifen, werden die Methoden des `document`-Objekts `getElementById`, `getElementsByName` und `getElementsByTagName` verwendet. Ausgehend davon können Sie die Attributknoten, Textknoten und weitere Element-Kindknoten eines Elements ansprechen. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<h1 id="Ueberschrift" align="center">Knoten in der <i>Baumstruktur</i></h1>
<script type="text/javascript">
<!--
  Elementknoten = document.getElementById("Ueberschrift");
  WertErsterKindknoten = Elementknoten.firstChild.nodeValue;
  document.write("Der Wert ihres ersten Kindknotens lautet: <b>" +
WertErsterKindknoten + "<\b>");
//-->
</script>
</body></html>
```



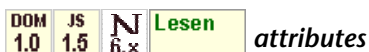
Die Beispieldatei enthält eine Überschrift erster Ordnung mit Text, von dem ein Teil wiederum als kursiv ausgezeichnet ist. In dem JavaScript, das unterhalb davon notiert ist, wird zunächst mit `document.getElementById("Ueberschrift")` (ohne weitere Eigenschaft oder Methode dahinter) auf das `h1`-Element der Überschrift zugegriffen. Der Rückgabewert von `getElementById()` ist das Knotenobjekt der Überschrift. Der Rückgabewert wird im Beispiel in der Variablen `Elementknoten` gespeichert. Diese Variable speichert also einen gültigen Knoten des Dokuments, und auf die Variable sind daher die Eigenschaften und Methoden des `node`-Objekts anwendbar. Im Beispiel wird mit `Elementknoten.firstChild.nodeValue` der Wert des ersten Kindknotens der Überschrift ermittelt. Dessen Wert wird schließlich mit `document.write()` ins Dokument geschrieben.

Die Verwendung von Variablen ist nicht zwingend erforderlich. Das obige Beispiel funktioniert genauso, wenn Sie notieren:

```
document.write("Der Wert ihres ersten Kindknotens lautet: <b>" +
document.getElementById("Ueberschrift").firstChild.nodeValue + "</b>");
```

Der geschriebene Wert lautet im Beispiel: Knoten in der ..., der erste Kindknoten der Überschrift ist also ihr Zeicheninhalt. Das Wort `Baumstruktur` gehört nicht dazu, da es ja durch ein `i`-Element ausgezeichnet ist, das selbst wieder einen eigenen, weiteren Kindknoten der Überschrift darstellt.

Die beiden Attribute, die im einleitenden Überschriften-Tag notiert sind, zählen übrigens nicht als Kindknoten. Das W3-Konsortium ist der Auffassung, dass Attribute hierarchisch gesehen keine »Unterobjekte« von Elementen sind, sondern »assoziierte Objekte«. Um auf Attributknoten zuzugreifen, bietet das `node`-Objekt eigene Eigenschaften und Methoden an. Zum Verständnis ist es jedoch wichtig, dass die Attribute eines Elements nicht als dessen Kindknoten betrachtet werden, weshalb etwa eine Objekteigenschaft wie `firstChild` die Attribute übergeht.



Speichert einen Array aus verfügbaren Attributen eines Elements. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body bgcolor="#FFFFCC" text="#000099">
<script type="text/javascript">
<!--
  document.write("Das body-Element hat <b>" +
document.getElementsByTagName('body')[0].attributes.length + "</b> Attribute");
//-->
</script>
</body></html>
```

Das Beispiel greift mit `document.getElementsByTagName('body')[0]` auf das `body`-Element der Datei zu und ermittelt über die Array-Grundeigenschaft `length` des `attributes`-Arrays die Anzahl der Attribute, die in dem Element definiert sind. Zur Kontrolle wird das Ergebnis mit `document.write` ins Dokument geschrieben. Im Beispiel ist 2 der Wert von `attributes.length`, da im einleitenden `<body>`-Tag zwei Attribute notiert sind.

Über den `attributes`-Array können Sie auch auf Attribute zugreifen. So liefert beispielsweise `document.getElementsByTagName('body')[0].attributes[0].nodeValue` den Wert `#FFFFCC`,

da `attributes[0]` das erste Attribut bezeichnet, und `nodeValue` den zugehörigen Wert speichert.

**Beachten Sie:** Der MS Internet Explorer ermittelt zwar einen Wert, jedoch nicht den richtigen. Im Beispiel wurde mit der Produktversion 5.5 statt 2 der Wert 94 ermittelt – vermutlich die Anzahl insgesamt verfügbarer Attribute für das `body`-Element, inklusive Event-Handler usw. Dies ist jedoch unbrauchbar.



Speichert einen Array aus verfügbaren Kindknoten eines Knotens. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p id="derText">Text mit <b>fetttem Text</b> und <u>unterstrichenem Text</u></p>
<script type="text/javascript">
<!--
var Anzahl = document.getElementById("derText").childNodes.length;
var Erster = document.getElementById("derText").childNodes[0].nodeValue;
document.write("Anzahl Kindknoten: <b>" + Anzahl + "</b><br>");
document.write("Wert des ersten Kindknotens: <b>" + Erster + "</b><br>");
//-->
</script>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit Zeicheninhalt und weiteren Elementen zur Formatierung. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementById("derText")` auf das `p`-Element zugegriffen. Über die Array-Grundeigenschaft `length` des `childNodes`-Arrays wird die Anzahl der Kindelemente ermittelt, die das `p`-Element hat. Der Rückgabewert wird in der Variablen `Anzahl` gespeichert. Über `childNodes[0]` wird auf das erste Kindelement zugegriffen. Dessen Inhalt wird mit `nodeValue` ermittelt. Zur Kontrolle schreibt das Script die Ergebnisse mit `document.write()` ins Dokument.

Die Anzahl der Kindelemente des `p`-Elements beträgt übrigens 4:

- Das erste Kindelement ist der Zeicheninhalt Text mit ,
- das zweite Kindelement das `b`-Element,
- das dritte Kindelement der Zeicheninhalt und ,
- und das vierte Kindelement das `u`-Element.

Wenn ein Knoten keine Kindknoten enthält, hat `childNodes` den Wert `null`.

**Beachten Sie:** Der Netscape 6.1 und der MS Internet Explorer 5.0 Macintosh Edition interpretieren im `body`-Element und in allen untergeordneten Knoten bereits einen Zeilenumbruch oder ein Leerzeichen im Quelltext zwischen Elementknoten als einen eigenen Kindknoten mit dem Namen `#text`. Auch HTML-Kommentare bilden eigene Knoten und werden als Knoten mit dem Namen `#comment` behandelt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<ul id="ersteListe"><li>erster Punkt</li><li>zweiter Punkt</li></ul>
<script type="text/javascript">
```

```

<!--
var Anzahl = document.getElementById("ersteListe").childNodes.length;
document.write("Erste Liste: Anzahl Kindknoten: <b>" + Anzahl + "</b><br>");
//-->
</script>

<ul id="zweiteListe">
<li>erster Punkt</li> <li>zweiter Punkt</li>
</ul>
<script type="text/javascript">
<!--
var Anzahl = document.getElementById("zweiteListe").childNodes.length;
document.write("Zweite Liste: Anzahl Kindknoten: <b>" + Anzahl + "</b><br>");
//-->
</script>
</body></html>

```

Im Beispiel sind zwei Listen definiert. Beide unterscheiden sich nur dahingehend, dass sich zwischen den einzelnen Elementen der ersten Liste keine Leerzeichen oder Zeilenumbrüche befinden. In der zweiten Liste dagegen wurden Leerzeichen und Zeilenumbrüche verwendet. Der Netscape 6.1 gibt für die erste Liste als Anzahl der Kindknoten 2 aus und für die zweite Liste 5. Die jeweiligen Leerzeichen und Zeilenumbrüche werden als eigene Kindknoten betrachtet.

Der MS Internet Explorer unter Windows ignoriert diese und erkennt erst dann einen Kindknoten mit dem Namen `#text`, wenn dieser ein entsprechendes Textzeichen oder ein erzwungenes Leerzeichen `&nbsp;` enthält.



Speichert Zeichendaten eines Knotens, sofern es sich um einen Textknoten handelt. Ein Beispiel:

```

<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
function Ausgeben() {
  alert(document.getElementById("Dokumentinhalt").firstChild.data);
}
//-->
</script>
</head><body id="Dokumentinhalt" onLoad="Ausgeben()">
Das ist Text, der einfach so im Dokument steht.
</body></html>

```

Das Beispiel enthält zwischen `<body>` und `</body>` nichts als reinen Text, also Zeichendaten. Nachdem das Dokument geladen ist (`onLoad`), wird die JavaScript-Funktion `Ausgeben()` aufgerufen. Diese greift mit `document.getElementById("Dokumentinhalt")` auf das `body`-Element zu. Mit `firstChild` wird auf das erste Kindelement zugegriffen, und mit `data` auf dessen Inhalt. Das erste (und einzige) Kindelement des `body`-Elements ist im Beispiel der Text. Dieser wird denn auch in dem Meldungsfenster angezeigt, das die Funktion `Ausgeben()` ausgibt.

Wenn Sie den Wert der Eigenschaft `data` ändern, wird der Zeicheninhalt des Elements einfach durch den neuen Inhalt ersetzt. Wenn Sie im obigen Beispiel also notieren würden:

```
document.getElementById("Dokumentinhalt").firstChild.data = "neuer Text";
```

dann würde anschließend der bisherige Text zwischen `<body>` und `</body>` dynamisch durch den neuen ersetzt.



*firstChild*

Speichert das Objekt des ersten Kindknotens eines Knotens. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<ul><li>erster Punkt</li><li>zweiter Punkt</li></ul>
<script type="text/javascript">
<!--
var ErsterPunkt = document.getElementsByTagName("ul")[0].firstChild;
document.write(ErsterPunkt.firstChild.data);
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Aufzählungsliste mit zwei Listenelementen. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("ul")[0]` auf das erste `ul`-Element im Dokument zugegriffen, mit `firstChild` wird dessen erster Kindknoten angesprochen. Im Beispiel ist dies das erste `li`-Element. In der Variablen `ErsterPunkt` ist also anschließend das Knotenobjekt des ersten `li`-Elements gespeichert, aber nicht sein Textinhalt. Dieser wird jedoch anschließend mit `document.write()` ausgegeben. Da der Textinhalt aus Sicht des `li`-Elementknotens wieder einen Unterknoten darstellt, kann mit `ErsterPunkt.firstChild.data` auf den Text des ersten Listenelementes zugegriffen werden.

Berücksichtigen Sie beim Nachvollziehen dieses Beispiels die Besonderheit des Netscape 6.1 und des MS Internet Explorer 5.0 Macintosh Edition im Umgang mit Kindknoten (siehe Abschnitt `childNodes`).



*lastChild*

Speichert das Objekt des letzten Kindknotens eines Knotens. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<ul><li>erster Punkt</li><li>zweiter Punkt</li></ul>
<script type="text/javascript">
<!--
var LetzterPunkt = document.getElementsByTagName("ul")[0].lastChild;
document.write(LetzterPunkt.firstChild.data + " und ");
document.write(LetzterPunkt.lastChild.data);
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Aufzählungsliste mit zwei Listenpunkten. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("ul")[0]` auf das erste `ul`-Element im Dokument zugegriffen. mit `lastChild` wird dessen letzter Kindknoten angesprochen. Im Beispiel ist dies das zweite und letzte `li`-Element. In der Variablen `LetzterPunkt` ist also anschließend das Knotenobjekt des zweiten `li`-Elements gespeichert, aber nicht sein Textinhalt. Dieser wird jedoch anschließend mit `document.write()` ausgegeben, und zwar gleich zweimal. Da der Textinhalt aus Sicht des `li`-Elementknotens wieder einen Unterknoten darstellt, und zwar den einzigen, kann er mit `LetzterPunkt.firstChild.data`, aber ebenso gut mit `LetzterPunkt.lastChild.data` angesprochen werden.

Berücksichtigen Sie beim Nachvollziehen dieses Beispiels die Besonderheit des Netscape 6.1 und des MS Internet Explorer 5.0 Macintosh Edition im Umgang mit Kindknoten (siehe `childNodes`).



### *nextSibling*

Speichert aus Sicht eines Knotens den unmittelbar nächstfolgenden Knoten im Strukturbaum. Wenn kein Knoten mehr folgt, wird `null` gespeichert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<ul>
<li>erster Punkt</li>
<li>zweiter Punkt</li>
</ul>
<script type="text/javascript">
<!--
document.write("Das ul-Element hat folgende Knoten unter sich:<br>");
var Knoten = document.getElementsByTagName("ul")[0].firstChild;
while (Knoten!=null) {
    document.write("Einen Knoten mit dem Namen <b>" + Knoten.nodeName + "<\b><br>");
    Knoten = Knoten.nextSibling;
}
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Aufzählungsliste mit zwei Listenpunkten. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("ul")[0].firstChild` auf das erste `ul`-Element im Dokument zugegriffen. In der nachfolgenden `while`-Schleife wird geprüft, ob der Knoten verschieden von `null` ist, und der `nodeName` des Knotens wird ausgegeben. Anschließend ist die Variable `Knoten` mit dem nachfolgenden Knoten (`Knoten = Knoten.nextSibling;`) belegt. Wenn der Strukturbaum durchlaufen ist, ist in der Eigenschaft `nextSibling` der Wert `null` gespeichert, was zum Abbruch der Schleife führt.

Die Eigenschaft `nextSibling` arbeitet jeweils den nächsten Knoten eines Strukturbaumes ab. Sie verfolgt jedoch nicht die Kindknoten, die in einem Knoten enthalten sein können.

**nodeName**

Speichert den Namen eines Knotens. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body><script id="dasScript" type="text/javascript">
<!--
Knoten = document.getElementById("dasScript");
var Knoten = document.getElementsByTagName("body")[0].firstChild;
document.write("Dieses Script-Element hat folgende Knotennamen: <b>" +
Knoten.nodeName + "</b>")
//-->
</script>
</body></html>
```

Das Beispiel enthält im sichtbaren Dokumentbereich nichts weiter als ein JavaScript, das den Knotennamen des eigenen script-Elements ausgibt, also SCRIPT. Dazu greift das Script mit `document.getElementsByTagName("body")[0].firstChild` auf das script-Element zu. Des-  
sen Knoten wird in der Variablen `Knoten` gespeichert. Mit `Knoten.nodeName` wird dann der Name dieses Knotens ermittelt. Elementknoten und Attributknoten haben Namen, Textknoten jedoch nicht. Beim Versuch, den Namen eines Textknotens zu ermitteln, wird der Wert `#text` gespeichert.

Berücksichtigen Sie beim Nachvollziehen dieses Beispiels die Besonderheit des Netscape 6.1 im Umgang mit Kindknoten (siehe `childNodes`).

**nodeType**

Speichert den Typ eines Knotens in Form einer Nummer. Das W3-Konsortium hat dazu folgende Zuordnungen festgelegt – einige davon sind XML-spezifisch:

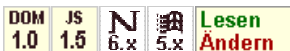
| Nummer | Knotentyp                         |
|--------|-----------------------------------|
| 1      | Elementknoten                     |
| 2      | Attributknoten                    |
| 3      | Textknoten                        |
| 4      | Knoten für CDATA-Bereich          |
| 5      | Knoten für Entity-Referenz        |
| 6      | Knoten für Entity                 |
| 7      | Knoten für Verarbeitungsanweisung |
| 8      | Knoten für Kommentar              |
| 9      | Dokumentknoten                    |
| 10     | Dokumenttyp-Knoten                |
| 11     | Dokumentfragment-Knoten           |
| 12     | Knoten für Notation               |

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p align="center">ein kleiner Text</p>
<script type="text/javascript">
<!--
var Element = document.getElementsByTagName("p")[0];
var Ausrichtung = Element.getAttributeNode("align");
alert(Ausrichtung.nodeType);
//-->
</script>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit einem Attribut `align=` zur Ausrichtung. Unterhalb des Textabsatzes ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementsByTagName("p")[0]` auf das `p`-Element zugegriffen. Mit `Element.getAttributeNode("align")` wird auf dessen Attributknoten zugegriffen. In der Variablen `Ausrichtung` steht anschließend das Objekt des Attributknotens. Ein Meldungsfenster gibt dann im Beispiel den Knotentyp dieses Knotens mit `Ausrichtung.nodeType` aus. Der Wert beträgt 2, da es sich um einen Attributknoten handelt.

**Beachten Sie:** Der MS Internet Explorer 5.x interpretiert die Eigenschaft `nodeType` zwar, in Version 5.5 aber nicht das obige Beispiel. Grund ist, dass er die Methode `getAttributeNode()` nicht unterstützt. Im MS Internet Explorer 6.0beta und im MS Internet Explorer 5.0 Macintosh Edition ist das Beispiel dagegen nachvollziehbar.

 **nodeValue**

Speichert den Wert oder Inhalt eines Knotens. Bei Textknoten ist dies der Text, bei Attributknoten der zugewiesene Attributwert. Bei Elementknoten hat diese Eigenschaft den Wert `null`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function TextAendern() {
  document.getElementById("derText").firstChild.nodeValue =
document.Formular.neuerText.value;
}
//-->
</script>
</head><body>
<p id="derText">Hier steht ein Text</p>
<form name="Formular" action="">
<input type="text" size="40" name="neuerText">
<input type="button" value=" OK " onClick="TextAendern()">
</form>
</body></html>
```

In dem Beispiel ist ein Textabsatz notiert und unterhalb davon ein Formular mit einem Eingabefeld und einem Klick-Button. Beim Anklicken des Buttons wird die Funktion `TextAendern()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("derText")` auf das `p`-Element zu, weiterhin mit `firstChild` auf den ersten Kindknoten dieses Elements, also den Textinhalt, und weist dessen Eigenschaft `nodeValue` den Inhalt aus dem Eingabefeld des Formulars zu.

**Beachten Sie:** Das Beispiel zeigt, dass `nodeValue` eine ähnliche Funktionalität hat wie die Eigenschaft `all.innerText` beim klassischen DHTML nach Microsoft-Syntax. Dennoch gibt es Unterschiede: Wenn beispielsweise notiert wäre:

```
<p>Text mit <b>fetttem Text</b></p>
```

dann würde `firstChild.nodeValue` aus Sicht des `p`-Elements nur den Wert `Text mit` liefern und auch nur diesen Teil ändern können, da dahinter durch das innere `b`-Element ein neuer Knoten beginnt.

Eine direkte Entsprechung zu `all.innerHTML` gibt es erst recht nicht im DOM. »Inneres HTML« muss im DOM mit Hilfe geeigneter Methoden wie `document.createElement()`, `document.createAttribute()` und `document.createTextNode()` erzeugt werden.

 **parentNode**

Speichert den Elternknoten eines Knotens. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<ul>
<li>ein Punkt</li>
<li>ein zweiter</li>
</ul>
<script type="text/javascript">
<!--
  alert(document.getElementsByTagName("li")[0].parentNode.parentNode.tagName);
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Aufzählungsliste. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird der Name des Großelternelements des ersten `li`-Elements in einem Meldungsfenster ausgegeben. Dazu wird mit `document.getElementsByTagName("li")[0]` auf das erste `li`-Element zugegriffen. Das erste `parentNode` dahinter greift auf dessen Elternelement zu, und das zweite `parentNode` auf das Elternelement des Elternelements. Von diesem Element wird mit `tagName` der Name des Elementes ermittelt und ausgegeben. Das Attribut `tagName` kennen alle Knoten der Art *Element*. Ausgegeben wird im Beispiel `BODY`.

**Beachten Sie:** Das vorliegende Beispiel wird auch von Opera 5.12 interpretiert.

 **previousSibling**

Speichert aus Sicht eines Knotens den unmittelbar vorhergehenden Knoten im Strukturbaum. Wenn kein Knoten vorausgeht, wird `null` gespeichert. Ein Beispiel:



```
<html><head><title>Test</title></head>
<body id="Dokumentinhalt"
onLoad="alert(document.getElementById('Dokumentinhalt').previousSibling.nodeName)">
</body></html>
```

Das Beispiel gibt, nachdem das Dokument geladen ist (onLoad), in einem Meldungsfenster den Knotennamen des Vorgängerknotens vom body-Element aus. Dazu wird mit `document.getElementById('Dokumentinhalt')` auf das body-Element zugegriffen. Ausgegeben wird dann HEAD, da das head-Element im Sinne des Strukturbauums das unmittelbare Vorgängerelement des body-Elements ist.

Berücksichtigen Sie bei der Verwendung dieser Eigenschaft die Besonderheit des Netscape 6.1 und des MS Internet Explorer 5.0 Macintosh Edition im Umgang mit Kindknoten (siehe `childNodes`).



### *appendChild()*

Hängt einen zuvor neu erzeugten Knoten in die bestehende Knotenstruktur ein, und zwar so, dass er als letztes Kindelement eines anzugebenden Knotens eingefügt wird. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body>
<ol id="Liste">
<li>Element</li>
</ol>
<script language="JavaScript" type="text/javascript">
<!--
document.getElementById("Liste").removeChild(document.getElementById("Liste").first
Child);

for(var i = 0; i < 10; i++) {
    var newLI = document.createElement("li");
    var liNummer = i + 1;
    var newLIText = document.createTextNode("Das ist Listeneintrag Nummer " +
liNummer);
    document.getElementById("Liste").appendChild(newLI);
    document.getElementsByTagName("li")[i].appendChild(newLIText);
}
//-->
</script>
</body></html>
```

Das Beispiel füllt eine nummerierte Liste automatisch mit Daten. Unmittelbar nach dem Einlesen der nummerierten Liste folgt ein JavaScript-Bereich. Zuerst wird mit der Methode `removeChild()` das vorhandene Listenelement entfernt. Anschließend ist eine `for`-Schleife notiert, die 10-mal durchlaufen wird. Bei jedem Schleifendurchgang wird zunächst mit `document.createElement()` ein neues Element vom Typ `li` erzeugt. Dann wird eine Variable `liNummer` auf einen Wert gesetzt, der um 1 höher ist als der des Schleifenzählers `i`. Diese Variable wird in der nachfolgenden Anweisung benutzt, bei der mit `document.createTextNode()` ein neuer Textknoten erzeugt wird. Anschließend folgen – immer noch innerhalb der `for`-Schleife – zwei `appendChild()`-Aufrufe. Die erste der Anweisungen greift mit `document.`

`getElementById("Liste")` auf das `ol`-Element zu und führt dazu, dass diesem ein neues Kindelement am Ende hinzugefügt wird. Angehängt wird der zuvor neu erzeugte Elementknoten `newLI`, der ja ein neues `li`-Element speichert. Beim zweiten Aufruf wird der Schleifenzähler `i` benutzt, um mit `document.getElementsByTagName("li")[i]` auf das gerade neu erzeugte `li`-Element zuzugreifen. Ihm wird mit `appendChild()` der zuvor erzeugte Textknoten als Kindelement hinzugefügt. Auf diese Weise füllt sich die Liste bei jedem Schleifendurchlauf um ein neues `li`-Element mitsamt Zeicheninhalt.

### **`appendData()`**

Fügt einem Textknoten oder dem Wert eines Attributknotens am Ende Daten hinzu, ohne die bestehenden Daten zu überschreiben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function ergaenzen() {
    var Rest = document.createTextNode("vollkommen!");
    document.getElementById("Absatz").firstChild.appendData(Rest.nodeValue);
}
//-->
</script>
</head>
<body>
<p id="Absatz">Ich bin ja so un</p>
<p><a href="javascript:ergaenzen()">un - was?</a></p>
</body></html>
```

Das Beispiel enthält einen Absatz mit nicht ganz sinnigem Text und unterhalb davon einen Verweis. Beim Anklicken des Verweises wird die Funktion `ergaenzen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion erzeugt zunächst mit `document.createTextNode()` einen neuen Textknoten. Anschließend greift sie mit `document.getElementById("Absatz").firstChild` auf den Knoten zu, der den Zeicheninhalt des Textabsatzes mit dem unfertigen Text darstellt, und fügt dort mit `appendData()` den Wert des neu erzeugten Textknotens (`Rest.nodeValue`) hinzu. Aus dem Text `ich bin ja so un` wird also `ich bin ja so unvollkommen!`

**Beachten Sie:** Der Internet Explorer 5.x unterstützt diese Methode noch nicht. Sie können sich damit behelfen, auf den Inhalt eines Textknotens zuzugreifen und dann mit `nodeValue += "Text"` Daten hinzuzufügen. Im MS Internet Explorer 6.0beta und im MS Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

### **`cloneNode()`**

Erstellt eine identische Kopie eines Knotens, je nach Wunsch mit oder ohne zugehörige Unterknotenstruktur. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body>
<span id="Dolly">Dolly </span>
<script type="text/javascript">
<!--
```

```
Dolly2 = document.getElementById("Dolly").cloneNode(true);
document.getElementById("Dolly").firstChild.nodeValue +=
Dolly2.firstChild.nodeValue;
//-->
</script>
</body></html>
```

Das Beispiel enthält einen in einem `span`-Element notierten Text `Dolly`. Unterhalb davon ist ein JavaScript-Bereich notiert. Dort wird mit `document.getElementById("Dolly")` auf das `span`-Element zugegriffen. Dieses wird mit `cloneNode()` kopiert, und der Rückgabewert wird in der Variablen `Dolly2` gespeichert. Der Rückgabewert ist eine identische Kopie des Elementknotens des `span`-Elements. Als Parameter wird der Methode im Beispiel `true` übergeben. Das bedeutet, dass auch der Textinhalt des Elements mitkopiert wird. Um nur das Element ohne den Inhalt zu klonen, müssen Sie `false` übergeben.

Im Beispiel wird anschließend mit `document.getElementById("Dolly").firstChild.nodeValue` der Inhalt des `span`-Elements angesprochen und mittels Zeichenkettenoperation um den Wert des Kindelements des Klons erweitert, auf den mit `Dolly2.firstChild.nodeValue` zugegriffen wird. Am Ende steht also `Dolly Dolly` als Inhalt in dem `span`-Element.

 **deleteData()**

Löscht Daten eines Textknotens oder den Wert eines Attributknotens. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function loeschen() {
  var AnzahlZeichen =
document.getElementsByTagName("p")[0].firstChild.nodeValue.length;
  document.getElementsByTagName("p")[0].firstChild.deleteData(0,AnzahlZeichen);
}
//-->
</script>
</head>
<body>
<p>Die Beredtsamkeit an sich ist die Tugend des Redens
und die Lust des Schweigens an seinem Gegenteil.
Die Beredtsamkeit ist also ...
<a href="javascript:loeschen()">och nee, besser nicht!</a></p>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit Text und einem Verweis. Beim Anklicken des Verweises wird die Funktion `loeschen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ermittelt zunächst durch Zugriff auf den Zeicheninhalt des Absatzes (`document.getElementsByTagName("p")[0].firstChild.nodeValue`) mit der Methode `string.length` dessen Zeichenanzahl. Der Wert wird für die folgende Anweisung benötigt. Dort wird wieder mit `document.getElementsByTagName("p")[0].firstChild` auf den Zeicheninhalt des Absatzes zugegriffen. Mit `deleteData()` wird der Inhalt gelöscht. Die Methode `deleteData()` erwartet zwei Parameter:

- das Zeichen, ab dem gelöscht werden soll (0 steht für »ab dem ersten Zeichen«),
- Anzahl Zeichen, die gelöscht werden sollen (im Beispiel wird dazu die Variable `AnzahlZeichen` übergeben, in der die zuvor ermittelte Anzahl der Zeichen im Textknoten gespeichert ist).

Der Verweis im Beispiel bleibt übrigens nach dem Löschen stehen, weil das `a`-Element des Verweises schon wieder einen neuen Kindknoten darstellt.

**Beachten Sie:** Der Internet Explorer 5.x unterstützt diese Methode noch nicht. Um die gesamten Zeichendaten eines Textknotens zu löschen, können Sie sich damit behelfen, auf den Knoten zuzugreifen und der Eigenschaft `nodeValue` den Wert `"` (leere Zeichenkette) zuzuweisen. Im MS Internet Explorer 6.0beta und im MS Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.



Ermittelt den Wert eines bestimmten Attributs in einem Element. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Anzeigen(attr) {
    alert(document.getElementsByTagName("body")[0].getAttribute(attr));
}
//-->
</script></head>
<body bgcolor="#FFFFCC" text="#E00000" link="#0000E0" alink="#000080"
vlink="#000000">
<a href="javascript:Anzeigen('bgcolor')">Hintergrundfarbe?</a><br>
<a href="javascript:Anzeigen('text')">Textfarbe?</a><br>
<a href="javascript:Anzeigen('link')">Linkfarbe noch nicht besuchte Seiten?</a><br>
<a href="javascript:Anzeigen('vlink')">Linkfarbe besuchte Seiten?</a><br>
<a href="javascript:Anzeigen('alink')">Linkfarbe aktivierte Links?</a>
</body></html>
```

Das Beispiel enthält mehrere Verweise. Alle Verweise rufen beim Anklicken die Funktion `Anzeigen()` auf, die im Dateikopf notiert ist. Übergeben wird der Funktion der gewünschte Attributname. Die Funktion greift mit `document.getElementsByTagName("body")[0]` auf das `body`-Element zu. Mit `getAttribute()` lassen sich dann Attributwerte des einleitenden `<body>`-Tags ermitteln. Die Methode erwartet den Namen des gewünschten Attributs und liefert dessen Wert zurück. Im Beispiel bekommt sie jeweils den der Funktion übergebenen Parameter `attr` weitergereicht.

**Beachten Sie:** Das obige Beispiel wird auch von Opera 5.12 interpretiert.



»Holt« einen Attributknoten. Liefert das Knotenobjekt des gewünschten Attributs zurück. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
```

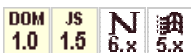
```

<!--
function anpassen() {
var  CSSKnoten = document.getElementsByTagName("h1")[0].getAttributeNode("style");
for(i = 0; i < document.getElementsByTagName("p").length; i++)
    document.getElementsByTagName("p")[i].setAttributeNode(CSSKnoten);
}
//-->
</script></head>
<body>
<h1 style="font-family:Tahoma; font-size:200%; color:red">Rot und Groß!</h1>
<p>ein Absatz</p>
<p>und noch einer</p>
<p>und einer mit einem <a href="javascript:anpassen()">Link zum Anpassen</a></p>
</body></html>

```

Das Beispiel enthält eine Überschrift mit einem `style`-Attribut sowie drei Textabsätze. Der letzte davon enthält einen Verweis, bei dessen Anklicken die Funktion `anpassen()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion holt sich mit `document.getElementsByTagName("h1")[0].getAttributeNode("style")` das Knotenobjekt des `style`-Elements aus der Überschrift. Der Rückgabewert, also das Attributknoten-Objekt, wird in der Variablen `CSSKnoten` gespeichert. Anschließend wird in einer `for`-Schleife auf alle `p`-Elemente des Dokuments zugegriffen. Mit `setAttributeNode()` wird dem jeweils aktuellen `p`-Element der gespeicherte `style`-Attributknoten hinzugefügt. Die `p`-Elemente »erben« auf diese Weise die CSS-Eigenschaften der Überschrift.

**Beachten Sie:** Der MS Internet Explorer 5.x interpretiert diese Methode noch nicht. Die Version 6.0 kennt diese Methode, jedoch wird sie nicht vollständig interpretiert.



### *hasChildNodes()*

Ermittelt, ob ein Knoten Kindknoten unter sich hat. Gibt den booleschen Wert `true` zurück, wenn ja, und `false`, wenn nein. Ein Beispiel:

```

<html><head><title>Test</title></head>
<body>
<p></p>
<p>mit Inhalt</p>
<p></p>
<p>mit Inhalt</p>
<script type="text/javascript">
<!--
for(i = 0; i < document.getElementsByTagName("p").length; i++)
    if(document.getElementsByTagName("p")[i].hasChildNodes())
        document.write((i+1) + ". p-Element hat einen Inhalt!<br>");
//-->
</script>
</body></html>

```

Das Beispiel enthält vier Textabsätze. Zwei davon haben einen Zeicheninhalt, zwei andere sind leer. Unterhalb der Textabsätze ist ein JavaScript notiert. Dieses greift in einer `for`-Schleife der Reihe nach mit `document.getElementsByTagName("p")` auf alle `p`-Elemente des Dokuments

zu. Dabei wird mit `if(document.getElementsByTagName("p")[i].hasChildNodes())` abgefragt, ob das jeweils aktuell in der Schleife behandelte `p`-Element Kindknoten hat. Wenn ja, wird mit `document.write()` ins Dokument geschrieben, beim wievielten Element ein Kindknoten gefunden wurde. Im Beispiel ist das beim zweiten und vierten Element der Fall, da diese beiden Elemente Textknoten enthalten.

### DOM 1.0 JS 1.5 N 6.x *insertBefore()*

Fügt innerhalb eines Knotens einen Kindknoten vor einem anderen Kindknoten ein. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body>
<p id="derText">Text <i id="derKursiveText">und mit kursivem Text</i></p>
<script type="text/javascript">
<!--
var neuB = document.createElement("b");
var neuBText = document.createTextNode("mit fettem Text ");
neuB.appendChild(neuBText);
document.getElementById("derText").insertBefore(neuB,
document.getElementById("derKursiveText"));
//-->
</script>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit Text und einem Kindelement `<i>...</i>`. Unterhalb des Absatzes ist ein JavaScript-Bereich notiert. Darin wird zunächst mit `document.createElement()` ein neues Element des Typs `b` für fetten Text erzeugt. Anschließend wird mit `document.createTextNode()` ein Textknoten erzeugt, der dann mit `appendChild()` als Inhalt des neu erzeugten `b`-Elements eingefügt wird. Der ganze Komplex aus `b`-Element mit Inhalt ist dann in der Variablen `neuB` gespeichert. Mit `document.getElementById("derText")` greift das Script sodann auf das `p`-Element zu und fügt mit `insertBefore()` das neue Element vor dem anderen Kindelement ein, das die Id `"derKursiveText"` hat. Am Ende lautet der komplette Inhalt des `p`-Elements dann:

```
Text <b>mit fettem Text </b><i>und mit kursivem Text</i>
```

Die Methode `insertBefore()` erwartet also zwei Parameter:

- das neu einzufügende Kindelement,
- ein Kindelement, vor dem das neue Kindelement eingefügt werden soll.

### DOM 1.0 JS 1.5 N 6.x *insertData()*

Fügt Zeichendaten in einem Textknoten ab einer bestimmten Zeichenposition ein. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function italiano() {
document.getElementsByTagName("h1")[0].firstChild.insertData(7, "duemilauno");
```

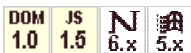
```

}
//-->
</script></head>
<body>
<h1>2001 = oder zweitausendeins</h1>
<a href="javascript:italiano()">sag es in italienisch!</a>
</body></html>

```

Das Beispiel enthält eine Überschrift erster Ordnung, in der noch irgendetwas fehlt an Text. Unterhalb davon ist ein Verweis notiert, bei dessen Anklicken die Funktion `italiano()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementsByTagName("h1")[0].firstChild` auf den Textknoten der Überschrift zu und fügt mit `insertData()` ab dem 8. Zeichen (Zeichenposition 7, zu zählen begonnen wird bei 0) die Zeichenkette `duemilauno` ein. Der dahinter stehende Text `oder zweitausendeins` wird – wie beim Einfügemodus in der Textverarbeitung – einfach nach hinten geschoben.

**Beachten Sie:** Der MS Internet Explorer 5.x interpretiert diese Methode noch nicht. Im MS Internet Explorer 6.0 und im MS Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.



### *removeAttribute()*

Löscht aus einem Element die Wertzuweisung an ein Attribut. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function wegMitAusrichtung() {
  document.getElementById("rechts").removeAttribute("align");
}
//-->
</script></head>
<body>
<p id="rechts" align="right"><a href="javascript:wegMitAusrichtung()">Weg mit der
Ausrichtung!</a></p>
</body></html>

```

Das Beispiel enthält einen mit `align="right"` rechtsbündig ausgerichteten Textabsatz, der einen Verweis enthält. Beim Anklicken des Verweises wird die Funktion `wegMitAusrichtung()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementById("rechts")` auf das `p`-Element zu und löscht mit `removeAttribute("align")` den Wert des `align`-Attributs. Der Absatz wird dadurch dynamisch neu ausgerichtet (per Voreinstellung linksbündig).

Wichtig ist aber, dass nicht das Attribut gelöscht wird, sondern nur dessen Wert. Im Beispiel ist der Effekt so, als wenn in HTML notiert wäre: `<p id="rechts" align="">...</p>`

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.




**removeAttributeNode()**

Löscht aus einem Element einen Attributknoten mitsamt Wertzuweisung. Ein Beispiel:

```
<html><head><title>Test</title></head>
<script language="JavaScript" type="text/javascript">
<!--
function wegMitAusrichtung() {

document.getElementById("rechts").removeAttributeNode(document.getElementById("rechts").attributes[1]);
}
//-->
</script>
<body>
<p id="rechts" align="right"><a href="javascript:wegMitAusrichtung()">Weg mit der
Ausrichtung!</a></p>
</body></html>
```

Das Beispiel tut das Gleiche wie das Beispiel bei der Beschreibung zu `removeAttribute()`. Der Unterschied ist nur, dass mit `removeAttributeNode()` das komplette Attribut gelöscht wird und nicht nur dessen Wertzuweisung. Als Parameter erwartet diese Methode das Objekt des gewünschten Attributknotens. Im Beispiel wird `document.getElementById("rechts").attributes[1]` übergeben, da das `align`-Attribut, das es zu entfernen gilt, das 2. Attribut im `attributes`-Array des `p`-Elements ist.

**Beachten Sie:** Der MS Internet Explorer 5.x interpretiert diese Methode noch nicht.





**removeChild()**

Löscht aus einem Element einen Kindknoten. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function wegMitEintrag() {
var Knoten=document.getElementsByTagName("ol")[0].firstChild;
verschwunden = document.getElementsByTagName("ol")[0].removeChild(Knoten);
alert(verschwunden.firstChild.nodeValue)
}
//-->
</script></head>
<body>
<ol start="1" type="I"><li>Element 1</li><li>Element 2</li><li>Element 3</li></ol>
<p><a href="javascript:wegMitEintrag()">Lösche das erste Element</a></p>
</body></html>
```

Das Beispiel enthält eine nummerierte Liste. Unterhalb davon ist ein Verweis notiert. Beim Anklicken des Verweises wird die Funktion `wegMitEintrag()` aufgerufen, die im Dateikopf notiert ist. In dieser Funktion wird auf den ersten Kindknoten mit `document.getElementsByTagName("ol")[0].firstChild` zugegriffen. Der Wert wird in der Variablen `Knoten` gespeichert. Mit



`removeChild(Knoten)` wird der Listenpunkt gelöscht. Dabei wird der Methode als Parameter der zu löschende Knoten übergeben. Die Methode `removeChild()` gibt als Rückgabewert das gelöschte Element zurück. Dieser Wert ist im Beispiel in der globalen Variablen `verschunden` gespeichert. Über diese Variable können Sie weiterhin auf die Eigenschaften des Knotens zugreifen. So wird im Beispiel noch einmal der Inhalt des ersten Kindknotens des gelöschten Elements ausgegeben.

Berücksichtigen Sie beim Nachvollziehen dieses Beispiels die Besonderheit des Netscape 6.1 und des MS Internet Explorer 5.0 Macintosh Edition im Umgang mit Kindknoten (siehe `childNodes`).

### *replaceChild()*

Ersetzt aus Sicht eines Knotens einen seiner Kindknoten durch einen anderen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function andereAntwort() {
  var Textknoten = document.createTextNode("Du darfst!");
  document.getElementById("z2").replaceChild(Textknoten,
document.getElementById("z2").firstChild);
}
//-->
</script></head>
<body>
<table border="1" bgcolor="#FFFFC0" width="30%">
<tr><td id="z1">Darf ich?</td><td id="z2">keine Ahnung!</td></tr>
</table>
<p><a href="javascript:andereAntwort()">andere Antwort!</a></p>
</body></html>
```

Das Beispiel enthält eine Tabelle mit zwei Zellen. Unterhalb der Tabelle ist ein Verweis notiert. Beim Anklicken des Verweises wird die Funktion `andereAntwort()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion erzeugt zunächst mit `document.createTextNode()` einen neuen Textknoten für die Antwort. Die Antwort wird in Form eines Knotenobjekts benötigt, weil die Methode `replaceChild()` sowohl den neuen Knoten als auch den zu ersetzenden in Form eines Knotenobjekts erwartet. Im Beispiel wird mit `document.getElementById("z2")` auf die zweite Tabellenzelle zugegriffen, um deren Inhalt, also den Textknoten, der ihren Kindknoten darstellt, durch den neu erzeugten Textknoten zu ersetzen. Dazu wird `replaceChild()` angewendet. Als erster Parameter wird der neue Knoten übergeben, als zweiter der zu ersetzende. Der neue Knoten ist im Beispiel in der Variablen `Textknoten` gespeichert. Auf den zu ersetzenden Knoten wird mit `document.getElementById("z2").firstChild` zugegriffen.

### *replaceData()*

Ersetzt Zeichendaten im Inhalt eines Elements oder in der Wertzuweisung an ein Attribut. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
```

```

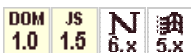
<!--
function andereAntwort() {
  var Text = "Du darfst!";
  var rd_Start = 0;
  var rd_Laenge = document.getElementById("z2").firstChild.nodeValue.length;
  document.getElementById("z2").firstChild.replaceData(rd_Start,rd_Laenge,Text);
}
//-->
</script></head>
<body>
<table border="1" bgcolor="#FFFC0" width="30%">
<tr><td id="z1">Darf ich?</td><td id="z2">keine Ahnung!</td></tr>
</table>
<p><a href="javascript:andereAntwort()">andere Antwort!</a></p>
</body></html>

```

Das Beispiel tut das Gleiche wie das Beispiel bei der Beschreibung zu `replaceChild()`. Der Unterschied ist nur, dass mit `replaceData()` auf Textebene gearbeitet wird und dass diese Methode auch gezieltes Ersetzen ab einer bestimmten Zeichenposition und für eine bestimmte Länge erlaubt. Als Parameter erwartet `replaceData()` zuerst die Startposition in der Zeichenkette, ab der ersetzt werden soll. Im Beispiel wird dazu die Variable `rd_Start` übergeben, der zuvor 0 zugewiesen wurde. Damit wird ab dem ersten Zeichen ersetzt. Als zweiter Parameter wird übergeben, wieviele Zeichen ersetzt werden sollen. Im Beispiel wird die Variable `rd_Laenge` übergeben. Dieser wurde zuvor mit `document.getElementById("z2").firstChild.nodeValue.length` die Zeichenanzahl des gesamten Textknotens der zweiten Tabellenzelle zugewiesen. Im Beispiel wird auf diese Weise der komplette Inhalt der zweiten Tabellenzelle ersetzt. Mit einer Angabe wie 0,1 würden Sie beispielsweise nur das erste Zeichen ersetzen.

Als dritten Parameter erwartet `replaceData()` eine Zeichenkette, mit der die zuvor im Textknoten markierte Zeichenkette ersetzt werden soll. Im Beispiel wird die Variable `Text` übergeben, der zuvor ein Wert zugewiesen wurde.

**Beachten Sie:** Der MS Internet Explorer 5.x interpretiert diese Methode noch nicht. Im MS Internet Explorer 6.0beta und im MS Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.



### *setAttribute()*

Setzt in einem Element einen Attributwert neu. Ist das Attribut bereits vorhanden, wird sein alter Wert durch den neuen ersetzt. Ist es noch nicht vorhanden, wird es neu angelegt und mit dem neuen Wert belegt. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function setzen() {
  document.getElementsByTagName("body")[0].setAttribute("bgColor",
document.Formular.bgcolor.value);
  document.getElementsByTagName("body")[0].setAttribute("text",
document.Formular.text.value);

```

```

}
//-->
</script></head>
<body bgcolor="#FFFFCC" text="#E00000">
<form name="Formular" action=""><pre>
Hintergrundfarbe:    <input type="text" name="bgcolor">
Textfarbe:           <input type="text" name="text">
Werte:               <input type="button" value="setzen" onClick="setzen()">
</pre></form>
</body></html>

```

Das Beispiel enthält ein Formular mit zwei Eingabefeldern, in denen der Anwender neue Werte für die beiden Attribute `bgcolor` und `text` des `body`-Elements eingeben kann. Beim Anklicken des Buttons unterhalb davon wird die Funktion `setzen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion greift mit `document.getElementsByTagName("body")[0]` auf das `body`-Element zu und weist ihm mit `setAttribute()` die neuen Attribute zu. Als erster Parameter wird der Name des zu erzeugenden oder zu ersetzenden Attributs übergeben, als zweiter Parameter der gewünschte Wert. Im Beispiel wird als zweiter Parameter jeweils der Wert aus den Formulareingabefeldern übergeben.

**Beachten Sie:** Der MS Internet Explorer ist empfindlich, was den ersten Parameter, also den gewünschten Attributnamen betrifft. Mit `bgColor` funktionierte das obige Beispiel, mit `bgcolor` dagegen nicht. Offenbar vermengt der Internet Explorer die erforderliche Syntax für das HTML-Elementobjekt `body` mit dem hier geforderten Wert. Im MS Internet Explorer 5.0 Macintosh Edition ist das Beispiel zwar nachvollziehbar, jedoch mit sehr seltsamen Ergebnissen.

#### *setAttributeNode()*

Fügt in ein Element einen neuen Attributknoten ein. Ist der Attributknoten bereits vorhanden, wird der alte Knoten durch den neuen ersetzt. Ist er noch nicht vorhanden, wird er neu angelegt. Ein Beispiel:

```

<html><head><title>Test</title></head>
<body>
<h1>Element ohne Eigenschaften?</h1>
<script type="text/javascript">
<!--
var Ausrichtung = document.createAttribute("align");
Ausrichtung.nodeValue = "center";
document.getElementsByTagName("h1")[0].setAttributeNode(Ausrichtung);
//-->
</script>
</body></html>

```

Das Beispiel enthält eine Überschrift erster Ordnung – ohne Attribute. Unterhalb davon ist ein JavaScript notiert. Dort wird zunächst mit `document.createAttribute()` ein neuer Attributknoten für ein Attribut namens `align` erzeugt. Der Knoten wird in der Variablen `Ausrichtung` gespeichert. Durch `Ausrichtung.nodeValue` lässt sich dem erzeugten Knoten dann ein Wert zuweisen. Mit `document.getElementsByTagName("h1")[0]` greift das Script schließlich auf das Überschriftenelement zu und weist ihm mit `setAttributeNode(Ausrichtung)` den zuvor erzeugten und in `Ausrichtung` gespeicherten Attributknoten zu.

**Beachten Sie:** Der MS Internet Explorer 5.x interpretiert diese Methode noch nicht. Im MS Internet Explorer 6.0beta und im MS Internet Explorer 5.0 Macintosh Edition wird die Methode dagegen unterstützt.

## 20.7 all

### Eigenschaften

|                |                                           |
|----------------|-------------------------------------------|
| className      | (Stylesheet-Klassenname eines Elements)   |
| dataFld        | (Datenfeld bei Datenanbindung)            |
| dataFormatAs   | (Datentyp bei Datenanbindung)             |
| dataPageSize   | (Anzahl Datensätze bei Datenanbindung)    |
| dataSrc        | (Datenquelle bei Datenanbindung)          |
| id             | (id-Name eines Elements)                  |
| innerHTML      | (Inhalt eines Elements als HTML)          |
| innerText      | (Inhalt eines Elements als Text)          |
| isTextEdit     | (Editierbarkeit eines Elements)           |
| lang           | (Sprache eines Elements)                  |
| language       | (Script-Sprache für ein Element)          |
| length         | (Anzahl Elemente)                         |
| offsetHeight   | (Höhe eines Elements)                     |
| offsetLeft     | (Linkswert der linken oberen Elementecke) |
| offsetParent   | (Elternelement-Position)                  |
| offsetTop      | (Obenwert der linken oberen Elementecke)  |
| offsetWidth    | (Breite eines Elements)                   |
| outerHTML      | (Elementinhalt plus äußeres HTML)         |
| outerText      | (Elementinhalt plus äußerem Text)         |
| parentElement  | (Elternelement)                           |
| parentTextEdit | (Editierbarkeit des Elternelements)       |
| recordNumber   | (Datensatznummer bei Datenanbindung)      |
| sourceIndex    | (wie vieltes Element)                     |
| tagName        | (HTML-Tag des Elements)                   |
| title          | (Titel des Elements)                      |

### Methoden

|                      |                                       |
|----------------------|---------------------------------------|
| click()              | (Element anklicken)                   |
| contains()           | (Zeichenkette in Element enthalten)   |
| getAttribute()       | (Attribut in einem Element ermitteln) |
| insertAdjacentHTML() | (Element einfügen)                    |
| insertAdjacentText() | (Text einfügen)                       |
| removeAttribute()    | (Attribut aus Element entfernen)      |
| scrollIntoView()     | (zu Element scrollen)                 |
| setAttribute()       | (Attribut eines Elements einfügen)    |

### **all: Allgemeines zur Verwendung**

Das Objekt `all`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, ist der Schlüssel zu Dynamischem HTML nach dem Ansatz des MS Internet Explorer ab Version 4.0. Mit Hilfe des `all`-Objekts haben Sie Script-Zugriff auf alle einzelnen Elemente und Inhalte einer HTML-Datei. Die meisten Eigenschaften können Sie lesen und ändern. Methoden des `all`-Objekts erlauben unter anderem das Einfügen oder Entfernen von HTML-Tags und von Angaben innerhalb eines HTML-Tags. Auf diese Weise ist der dynamische Zugriff auf alle Bestandteile einer Datei möglich.

Das `all`-Objekt gehört nicht zum offiziellen JavaScript-Sprachstandard. Es wurde von Microsoft für den MS Internet Explorer 4.0 implementiert. Das `all`-Objekt funktioniert zwar auch prima innerhalb von Script-Bereichen, die mit »JavaScript« ausgezeichnet sind, doch es ist bislang eigentlich nur Bestandteil von JScript, dem Microsoft-Derivat von JavaScript.

Mit den neuen Browser-Generationen und dem Document Object Model (DOM) wird das `all`-Objekt durch die neuen HTML-Elementobjekte und das `node`-Objekt verdrängt. Es hat also keine Zukunft mehr und sollte allenfalls noch aus Gründen der Rückwärtskompatibilität eingesetzt werden.

Eine zusammenhängende Einführung in das Arbeiten mit dem `all`-Objekt finden Sie im Kapitel 21.3, *Das ältere DHTML-Modell von Microsoft*.



### **HTML-Elemente ansprechen**

Das `all`-Objekt behandelt alle HTML-Tags, die in einer HTML-Datei notiert sind, als Kette von Elementen. Sie können jedes einzelne HTML-Tag in einer Datei ansprechen. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function Aendern() {
    document.all.tags("h1")[0].innerText = "Anderer Inhalt";
}
//-->
</script>
</head>
<body>
<h1>eine Überschrift</h1>
<a href="javascript:Aendern()">Aendern</a>
</body></html>
```

Das Beispiel enthält eine Überschrift und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Aendern()` aufgerufen. Diese Funktion ändert den Text der Überschrift dynamisch in "Anderer Inhalt".

Um ein bestimmtes HTML-Tag anzusprechen, notieren Sie `document.all.tags`. Dahinter folgt, in runden Klammern und in Anführungszeichen, der Name des HTML-Tags, auf das Sie zugreifen möchten. Im Beispiel soll auf ein `<h1>`-Tag zugegriffen werden, also muss ("`h1`") notiert werden. Damit haben Sie eine Art Zwischenobjekt, nämlich eine Kette aus allen `<h1>`-

Elementen der Datei. Nun müssen Sie noch angeben, welches Sie meinen. Im Beispiel gibt es zwar nur eine `<h1>`-Überschrift, aber es könnte ja auch mehrere davon geben. Dazu notieren Sie in eckigen Klammern, das wievielte HTML-Tag vom angegebenen Typ Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h., das erste Element sprechen Sie mit `[0]` an, das zweite mit `[1]` usw. Beim Zählen gilt die Reihenfolge, in der die HTML-Tags des ausgewählten Typs in der Datei notiert sind.

Nachdem Sie ein bestimmtes, eindeutiges HTML-Tag angesprochen haben, können Sie dahinter eine der Eigenschaften oder Methoden des `all`-Objekts auf dieses HTML-Tag anwenden.



### HTML-Elemente mit Namen ansprechen

Sie können jedes HTML-Element in einem Script auch mit einem Namen ansprechen. Voraussetzung ist, dass das HTML-Element einen Namen durch die Attribute `id=` bzw. `name=` erhalten hat. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
<script type="text/javascript">
<!--
function MachPzuH() {
  document.all.MeinElement.outerHTML = "<h1>" +
  document.all["MeinElement"].innerText + "<\h1>";
}
//-->
</script>
</head>
<body>
<p id="MeinElement">Ein Element mit Text</p>
<a href="javascript:MachPzuH()">Mach P zu H</a>
</body></html>
```

Das Beispiel enthält einen Absatz und einen Verweis. Beim Anklicken des Verweises wird die Funktion `MachPzuH()` aufgerufen. Diese Funktion verwandelt den Absatz dynamisch in eine Überschrift 1. Ordnung.

Um ein HTML-Element mit seinem Namen anzusprechen, müssen Sie im einleitenden Tag des Elements das Attribut `id=` notieren und dahinter einen Namen vergeben. Im Beispiel ist das der Name `"MeinElement"`. Dann können Sie dieses Element mit `document.all.ElementName`, im Beispiel also mit `document.all.MeinElement`, ansprechen.

Alternativ können Sie, wie im zweiten Teil des Beispiels, das Element mit `document.all["MeinElement"]` ansprechen. Über diese Art des Ansprechens eines Elementes können Sie auch Variablen übergeben. Dazu notieren Sie einfach den Variablennamen ohne Anführungszeichen anstatt eines festen Wertes, z.B. `document.all[Variablenname]`. Voraussetzung ist jedoch, dass die Variable auch existiert.

**className**

Speichert die Stylesheet-Klasse, zu der ein Element gehört. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
p.normal { color:black }
p.spezial { color:red }
-->
</style>
</head><body>
<p class="normal" id="derAbsatz">Text</p>
<script type="text/javascript">
<!--
document.write("<br>Klasse davor: " + document.all.derAbsatz.className);
document.all.derAbsatz.className = "spezial";
document.write("<br>Klasse danach: " + document.all.derAbsatz.className);
//-->
</script>
</body></html>
```

Im Beispiel werden im Dateikopf mit Hilfe von CSS Stylesheets zwei Klassen für das HTML-Tag `<p>` definiert: eine Klasse `normal` und eine Klasse `spezial`. Im Dateikörper wird ein Absatz definiert, der zunächst die Klasse `normal` zugewiesen bekommt. In einem Script wird der Klassenname des Absatzes zunächst mit `document.write()` zur Kontrolle in die Datei geschrieben. Dann wird dem Absatz die andere definierte Klasse zugewiesen. Dabei übernimmt der Absatz auch dynamisch alle Stylesheet-Eigenschaften der neuen Klasse. Anschließend wird der neue Klassenname zur Kontrolle auch noch mal in die Datei geschrieben.

**dataFld**

Speichert den Namen eines Datenfeldes, das zur Ausgabe eines Datensatzes gehört. Dies bezieht sich auf das Konzept der Datenanbindung. Zur Einführung in das Thema lesen Sie das Kapitel 21.5 über *Datenanbindung (Microsoft)*. Ein Beispiel:

```
<tr>
<td><b>Vorname:</b></td>
<td><span id="Test" dataFld="vorname">
</span><input type="button" value="Test"
onClick="alert(document.all.Test.dataFld)">
</td>
</tr>
```

Das Beispiel stellt eine Tabellenzeile dar, in der ein Datenfeld ausgegeben wird (die übrigen Befehle zur Datenanbindung fehlen hier). Zu Testzwecken ist ein Button eingefügt, bei dessen Anklicken der Name des Datenfeldes mit `alert()` ausgegeben wird. Es ist der Feldname, der in der HTML-Angabe `dataFld=` steht. Wenn Sie den Wert dynamisch ändern, können Sie in der gleichen Tabellenzeile beispielsweise ein anderes Feld ausgeben.

**dataFormatAs**

Speichert den Ausgabetypp eines Datenfeldes, das zur Ausgabe eines Datensatzes gehört. Dies bezieht sich auf das Konzept der Datenanbindung. Zur Einführung in das Thema lesen Sie das Kapitel 21.5 über *Datenanbindung (Microsoft)*. Ein Beispiel:

```
<tr>
<td><b>Vorname:</b></td>
<td><span id="Test" datafld="vorname" dataformatas="HTML">
</span><input type="button" value="Test"
onClick="alert(document.all.Test.dataFormatAs)">
</td>
</tr>
```

Das Beispiel stellt eine Tabellenzeile dar, in der ein Datenfeld ausgegeben wird (die übrigen Befehle zur Datenanbindung fehlen hier). Zu Testzwecken ist ein Button eingefügt, bei dessen Anklicken der Ausgabetypp des Datenfeldes mit `alert()` angezeigt wird. Da dieser im Ausgabebereich mit »HTML« festgelegt wurde, wird dieser Wert ausgegeben.

**dataPageSize**

Speichert, wie viele Datensätze gleichzeitig ausgegeben werden sollen. Dies bezieht sich auf das Konzept der Datenanbindung. Zur Einführung in das Thema lesen Sie das Kapitel 21.5 über *Datenanbindung (Microsoft)*. Ein Beispiel:

```
<input type="button" value="Test"
onClick="document.all.Anzeigetabelle.dataPageSize = 5">
<table id="Anzeigetabelle" datasrc="#Adressen" datapagesize="1">
...usw. Daten ...
</table>
```

Das Beispiel deutet eine Tabelle an, die zur Ausgabe von Daten aus einer Datenanbindung gedacht ist (die übrigen Befehle zur Datenanbindung fehlen hier). Da im einleitenden `<table>`-Tag die Angabe `datapagesize="1"` steht, wird in der Tabelle immer nur ein Datensatz gleichzeitig ausgegeben. Zu Testzwecken ist ein Button eingefügt, bei dessen Anklicken sich der Wert auf 5 ändert. Danach werden immer 5 Datensätze gleichzeitig angezeigt.

**dataSrc**

Speichert den Verweis auf das `<object>`-Tag, in dem eine Datenanbindung definiert wird. Dies bezieht sich auf das Konzept der Datenanbindung. Zur Einführung in das Thema lesen Sie das Kapitel 21.5 über *Datenanbindung (Microsoft)*. Ein Beispiel:

```
<input type="button" value="Test"
onClick="alert(document.all.Anzeigetabelle.dataSrc)">
<table id="Anzeigetabelle" datasrc="#Adressen" datapagesize="1">
...usw. Daten ...
</table>
```

Das Beispiel deutet eine Tabelle an, die zur Ausgabe von Daten aus einer Datenanbindung gedacht ist (die übrigen Befehle zur Datenanbindung fehlen hier). Zu Testzwecken ist ein Button eingefügt, bei dessen Anklicken der Verweis auf die Datenquelle ausgegeben wird. Da dieser im



Ausgabebereich mit "#Adressen" festgelegt wurde, wird dieser Wert ausgegeben. Durch Ändern dieser Objekteigenschaft können Sie dynamisch die Datenquelle wechseln. Voraussetzung ist, dass für den neuen Verweis auf die Datenquelle ein entsprechendes `<object>`-Tag für die Datenquelle definiert ist.



*id*

Speichert den Namen eines Elements. Im Beispiel wird ein Absatz definiert, der mit `id=` den Namen "meinAbsatz" zugewiesen bekommt. In einem Script wird der vergebene Name mit `document.write()` zur Kontrolle in die Datei geschrieben:

```
<html><head><title>Test</title>
</head><body>
<p id="meinAbsatz">Text</p>
<script type="text/javascript">
<!--
  document.write("<br>Name des Absatzes: " + document.all.meinAbsatz.id);
//-->
</script>
</body></html>
```



*innerHTML*

Speichert den Inhalt eines HTML-Elements. Wenn Sie beim dynamischen Ändern des gespeicherten Inhalts HTML-Tags notieren, werden diese bei der Aktualisierung des Elementinhalts interpretiert. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Neu = "neuer <b>fetter</b> Text";
function Aendern() { document.all.meinAbsatz.innerHTML = Neu; }
//-->
</script>
</head><body>
<p id="meinAbsatz">Text</p>
<a href="javascript:Aendern()">Anderer Text</a>
</body></html>
```

Das Beispiel enthält einen Textabsatz und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Aendern()` aufgerufen. Diese Funktion weist dem Absatz mit der `id="meinAbsatz"` für die Eigenschaft `innerHTML` den Wert der zuvor definierten Variablen `Neu` zu. Der Inhalt des Absatzes ändert sich dann dynamisch und berücksichtigt dabei auch die HTML-Formatierung `<b>...</b>` beim neuen Inhalt des Elements.

Die Eigenschaft `innerHTML` sollten Sie nicht direkt beim Einlesen der HTML-Datei anwenden, sondern immer erst abhängig von Aktionen wie Verweisklicks oder Button-Klicks oder mit einem `setTimeout()` von einigen Sekunden davor. Bei Anwendung direkt beim Einlesen der Datei meldet der MS Internet Explorer 4.0 einen Laufzeitfehler.



Speichert den Textinhalt eines HTML-Elements. Sie können den Inhalt dynamisch ändern. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Neu = "neuer Text";
function Aendern() { document.all.meinAbsatz.innerText = Neu; }
//-->
</script>
</head><body>
<p id="meinAbsatz">Text</p>
<a href="javascript:Aendern()">Anderer Text</a>
</body></html>
```

Das Beispiel enthält einen Textabsatz und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Aendern()` aufgerufen. Diese Funktion weist dem Absatz mit der `id="meinAbsatz"` für die Eigenschaft `innerText` den Wert der zuvor definierten Variablen `Neu` zu. Der Inhalt des Absatzes ändert sich dann dynamisch.

Die Eigenschaft `innerText` sollten Sie nicht direkt beim Einlesen der HTML-Datei anwenden, sondern immer erst abhängig von Aktionen wie Verweisklicks oder Button-Klicks oder mit einem `setTimeout()` von einigen Sekunden davor. Bei Anwendung direkt beim Einlesen der Datei meldet der MS Internet Explorer 4.0 einen Laufzeitfehler.



Speichert die Information, ob ein Element editierbar ist oder nicht. Wenn editierbar, hat die Eigenschaft den Wert `true`, wenn nicht editierbar, hat sie den Wert `false`. Derzeit sind nur bestimmte Formularelemente editierbar. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Ausgabe() {
  var Ausgabertext = "";
  for(var i=0;i<document.Testform.length;++i)
    Ausgabertext = Ausgabertext + "<br>" + document.all.tags("input")[i].isTextEdit;
  document.all.Ausgabe.innerHTML = Ausgabertext;
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<input type="text" size="30"><br>
<input type="radio"><br>
<input type="button" value="Test" onClick="Ausgabe()">
</form>
<p id="Ausgabe"></p>
</body></html>
```

Das Beispiel definiert ein Formular mit einigen verschiedenen Elementen. Das letzte Element ist ein Button. Beim Anklicken des Buttons wird die Funktion `Ausgabe()` aufgerufen. Diese Funktion ermittelt in einer Schleife für jedes Element des Formulars, ob es editierbar ist oder nicht. Die Werte werden in einer Variablen `Ausgabertext` gesammelt und HTML-formatiert. Am Ende wird dem Textabsatz, der in der Datei am Ende definiert ist, mit der Eigenschaft `innerHTML` der gespeicherte Wert von `Ausgabertext` zugewiesen. Auf diese Weise steht hinterher in der Datei, welche der Formularfelder als editierbar gelten und welche nicht.



Speichert die Sprache eines HTML-Tags, wie sie im Attribut `lang=` definierbar ist. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p id="Absatz" lang="it">mi chiamo Stefan</p>
<script type="text/javascript">
<!--
  alert(document.all.Absatz.lang);
//-->
</script>
</body></html>
```

Im Beispiel wird ein Textabsatz definiert, dem mit `lang=` die Sprache Italienisch (`it`) zugewiesen wird. Unterhalb davon steht ein kleines JavaScript, das mit `alert()` den vergebenen Wert für die Sprache ausgibt.



Speichert, welche Script-Sprache in einem aktuellen Script verwendet wird. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
  alert(alert(document.all.tags("script")[0].language);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel gibt mit `alert()` aus, welche Script-Sprache für die Ausgabe bewirkt hat. Das ist derjenige Wert, der im `<script>`-Tag beim Attribut `language=` vergeben wurde. Da der MS Internet Explorer mehrere Script-Sprachen beherrscht, kann dieser Befehl in manchen Fällen für Gewissheit sorgen, etwa wenn man ihn in eine If-Abfrage einbettet.



Speichert die Anzahl HTML-Tags einer HTML-Datei. Tags, die aus einem einleitenden und einem abschließenden Tag bestehen, werden als ein Tag gezählt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p>Ein Absatz</p>
```

```

<p>Noch ein Absatz</p>
<script type="text/javascript">
<!--
alert("Diese Datei hat " + document.all.length + " HTML-Tags");
//-->
</script>
</body></html>

```

Das Beispiel zeigt eine vollständige HTML-Datei mit Grundgerüst, zwei Textabsätzen und einem Script. Innerhalb des Scripts wird mit `alert()` ausgegeben, wie viele HTML-Tags die Datei enthält.



Lesen

***offsetHeight***

Speichert die Höhe eines Elements. Das Beispiel gibt mit `alert()` die reale Höhe des `<body>`-Tags der Datei aus. Das Ergebnis ist die tatsächliche Anzeigehöhe des Fensters, in dem das Dokument angezeigt wird. Die Eigenschaft lässt sich aber ebenso gut auf HTML-Elemente innerhalb der angezeigten Inhalte anwenden:

```

<html><head><title>Test</title>
</head><body id="DieseDatei">
<script type="text/javascript">
<!--
  alert(document.all.DieseDatei.offsetHeight);
//-->
</script>
</body></html>

```



Lesen

***offsetLeft***

Speichert den Abstand eines Elementes zum linken Rand des in der Eigenschaft `offsetParent` gespeicherten Offset-Elternelementes. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<div id="Bereich" style="padding:20px">
<p id="Absatz">Hier etwas Text</p>
</div>
<script language="JavaScript" type="text/javascript">
<!--
  alert(document.all.Bereich.offsetLeft);
  alert(document.all.Absatz.offsetLeft);
//-->
</script>
</body></html>

```

Im Beispiel wird ein `<div>`-Tag mit einem `<p>`-Tag innerhalb davon definiert. Damit ein wenig innerer Abstand in die Sache kommt, wird beim `<div>`-Tag mit Hilfe einer Stylesheet-Angabe ein Innenabstand definiert. Unterhalb dieser Befehle steht ein JavaScript, das die Abstände der beiden Elemente zum jeweiligen linken Rand des Offset-Elternelementes mit `alert()` ausgibt. Das Offset-Elternelement des Elementes `Bereich` ist das `Body`-Element. Für dieses Element wird

der Abstand zwischen Body und Bereich in der Eigenschaft `offsetLeft` gespeichert. Das Offset-Elternelement des Elementes Absatz ist das Element Bereich selbst. In der Eigenschaft `offsetLeft` ist jetzt der Abstand vom linken Rand vom Element Bereich bis zum Element gespeichert.

**Beachten Sie:** Besitzt ein Element einen Innenabstand (`padding`), so ist im MS Internet Explorer 5.0 die Eigenschaft `offsetLeft` fehlerhaft, da der gesetzte Innenabstand in die Angabe mit einfließt. Das gilt auch für `Padding`-Angaben, die in einem inneren Element definiert sind.



Lesen

### *offsetParent*

Speichert dasjenige Elternelement eines Elements, von dessen Positionierung die Positionierung des Elements abhängt. `offsetParent` ist dabei eine Art Zeiger auf das übergeordnete Element. Hinter der Eigenschaft lassen sich wiederum alle Eigenschaften und Methoden des `all`-Objekts notieren. Diese gelten dann für das Element, auf das der Zeiger zeigt. Existiert kein Offset-Elternelement, so hat die Eigenschaft `offsetParent` den Wert `null`. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<table><tr><td><div>
<a><b id="Fett">Hier etwas Text</b></a>
</div></td></tr></table>

<script type="text/javascript">
<!--

var Eltern=document.all.Fett.offsetParent;
while (Eltern) {
document.write(Eltern.tagName+"<br>");
Eltern=Eltern.offsetParent;
}
//-->
</script>
</body></html>
```

Im Beispiel ist eine Tabelle notiert, in deren einziger Zelle weitere Elemente enthalten sind. Das innerste Element ist das Element `<b>..</b>` mit dem `id`-Namen `Fett`. Im nachfolgenden JavaScript-Bereich wird der Variablen `Eltern` das Offset-Elternelement des Elementes `Fett` mit Hilfe von `offsetParent` zugewiesen. Die nachfolgende `while`-Schleife gibt zu jedem Offset-Elternelement den Namen des Tags aus. Anschließend wird der Variablen `Eltern` das Offset-Elternelement des gerade angesprochenen Elementes zugewiesen. Die Schleife bricht ab, wenn kein Offset-Elternelement mehr existiert. So wie im Beispiel die `all`-Objekteigenschaft `tagName` auf `offsetParent` angewendet wird, lassen sich auch alle anderen Eigenschaften und Methoden des `all`-Objekts auf das Element anwenden.

Im Unterschied zur Eigenschaft `parentElement` wird in der Eigenschaft `offsetParent` immer dasjenige Element gespeichert, das die Position des Elements im Fenster festlegt.

**Beachten Sie:** Im MS Internet Explorer 4.0 ist das Offset-Elternelement von einer Tabellenzelle immer die Tabellenzeile. Ab dem MS Internet Explorer 5.0 wird die Tabelle selbst als Offset-Elternelement einer Zelle angesehen.



Lesen

**offsetTop**

Speichert den Abstand eines Elements zum oberen Rand des in der Eigenschaft `offsetParent` gespeicherten Offset-Elternelements. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<div id="Bereich" style="padding:20px">
<p id="Absatz">Hier etwas Text</p>
</div>
<script type="text/javascript">
<!--
  alert(document.all.Bereich.offsetTop);
  alert(document.all.Absatz.offsetTop);
//-->
</script>
</body></html>
```

Im Beispiel wird ein `<div>`-Tag mit einem `<p>`-Tag innerhalb davon definiert. Damit ein wenig innerer Abstand in die Sache kommt, wird beim `<div>`-Tag mit Hilfe einer Stylesheet-Angabe ein Innenabstand definiert. Unterhalb dieser Elemente ist ein JavaScript notiert, das mit `alert()` die Abstände der beiden Elemente zum oberen Rand des Offset-Elternelementes ausgibt.

**Beachten Sie:** Im MS Internet Explorer 5.0 wird in einem untergeordneten Element eine bestehende Padding-Angabe des übergeordneten Elementes nicht berücksichtigt, sofern sich kein Element vor dem abgefragten Element befindet.



Lesen

**offsetWidth**

Speichert die Breite eines Elements. Das Beispiel gibt mit `alert()` die reale Breite des `<body>`-Tags der Datei aus. Das Ergebnis ist die tatsächliche Anzeigebreite des Fensters, in dem das Dokument angezeigt wird. Die Eigenschaft lässt sich aber ebenso gut auf HTML-Elemente innerhalb der angezeigten Inhalte anwenden.

```
<html><head><title>Test</title>
</head><body id="DieseDatei">
<script type="text/javascript">
<!--
  alert(document.all.DieseDatei.offsetWidth);
//-->
</script>
</body></html>
```



Lesen

Ändern

**outerHTML**

Speichert den Inhalt eines HTML-Tags plus das Anfangs- und End-Tag mit allen Angaben. Ein Beispiel:

```
<html>
<head>
<title>Test</title>
```

```

</head>
<body>
<a id="Verweis" href="javascript:alert(this.Verweis.outerHTML)">Verweis</a>
</body></html>

```

Das Beispiel enthält einen Verweis, der beim Anklicken seinen eigenen vollständigen HTML-Konstrukt mit `alert()` ausgibt. Die Eigenschaft `outerHTML` sollten Sie nicht direkt beim Einlesen der HTML-Datei anwenden, sondern immer erst abhängig von Aktionen wie Verweisklicks oder Button-Klicks oder mit einem `setTimeout()` von einigen Sekunden davor. Bei Anwendung direkt beim Einlesen der Datei meldet der MS Internet Explorer 4.0 einen Laufzeitfehler.



Speichert den gleichen Wert wie `innerText`, kann jedoch beim Ändern umgebende HTML-Tags entfernen und durch Text ersetzen. Ein Beispiel:

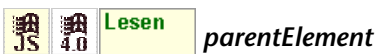
```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
  document.all.fett.outerText = document.all.fett.innerText;
}
//-->
</script>
</head><body>
<p>Text mit <b id="fett" onclick="Test()">fetter Schrift</b></p>
</body></html>

```

Das Beispiel enthält einen Text mit einem als fett markierten Teil. Beim Anklicken des fetten Teils wird die Funktion `Test()` aufgerufen. Diese Funktion ersetzt den Wert von `outerText` des fetten Elements durch den Wert von `innerText` des gleichen Elements. Der Effekt ist, dass die Formatierung mit `<b>...</b>` verloren geht, weil `outerText` intern die umgebende HTML-Formatierung mit speichert, `innerText` aber nicht.

Die Eigenschaft `outerText` sollten Sie nicht direkt beim Einlesen der HTML-Datei anwenden, sondern immer erst abhängig von Aktionen wie Verweisklicks oder Button-Klicks oder mit einem `setTimeout()` von einigen Sekunden davor. Bei Anwendung direkt beim Einlesen der Datei meldet der MS Internet Explorer 4.0 einen Laufzeitfehler.



Speichert das Elternelement eines Elements. `parentElement` ist dabei eine Art Zeiger auf das übergeordnete Element. Hinter der Eigenschaft lassen sich wiederum alle Eigenschaften und Methoden des `all`-Objekts notieren. Diese gelten dann für das Element, auf das der Zeiger zeigt. Existiert kein Elternelement, so hat die Eigenschaft `parentElement` den Wert `null`. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<table><tr><td><div>
<a><b id="Fett">Hier etwas Text</b></a>
</div></td></tr></table>

```

```

<script type="text/javascript">
<!--
  var Eltern=document.all.Fett.parentElement;
  while (Eltern) {
    document.write(Elttern.tagName+"<br>");
    Eltern=Eltern.parentElement;
  }
//-->
</script>
</body></html>

```

Im Beispiel ist eine Tabelle notiert, in deren einziger Zelle weitere Elemente enthalten sind. Das innerste Element ist das Element `<b>..</b>` mit dem Namen `Fett`. Im JavaScript-Bereich wird der Variablen `Eltern` das Elternelement des Elementes `Fett` mit der Eigenschaft `parentElement` zugewiesen. Die nachfolgende `while`-Schleife gibt von jedem Elternelement den Namen des Tags aus. Anschließend wird der Variablen `Eltern` das Elternelement des gerade angesprochenen Elementes zugewiesen. Die Schleife bricht ab, wenn kein Elternelement mehr existiert. So wie im Beispiel die `all`-Objekteigenschaft `tagName` auf `parentElement` angewendet wird, lassen sich auch alle anderen Eigenschaften und Methoden des `all`-Objekts auf das Element anwenden.

Im Unterschied zur Eigenschaft `offsetParent` wird in der Eigenschaft `parentElement` immer das unmittelbar übergeordnete Element gespeichert.



Lesen

### *parentTextEdit*

Speichert das nächsthöhere Element in der Elementhierarchie, das Editieren von Text erlaubt. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
  alert(document.all.Verweis.parentElement.tagName);
}
//-->
</script>
</head><body>
<a id="Verweis" href="javascript:Test()">Editieren?</a>
</body></html>

```

Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion gibt mit `alert()` den Namen desjenigen HTML-Tags aus, das aus Sicht des Verweises das nächsthöhere Element darstellt, das Editieren von Text erlaubt. Im Beispiel ist dies das `<body>`-Tag. Der Inhalt dieses Tags ist zwar nicht editierbar, aber es ist dasjenige nächsthöhere Tag, unterhalb dessen editierbare Elemente wie Formular-Eingabefelder notiert werden können.

**Beachten Sie:** MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.





Lesen

**recordNumber**

Speichert die Information, der wievielte Datensatz angezeigt wird. Dies bezieht sich auf das Konzept der Datenanbindung. Zur Einführung in das Thema lesen Sie das Kapitel 21.5 über *Datenanbindung (Microsoft)*. Ein Beispiel:

```
<tr>
<td><b>Vorname:</b></td>
<td><span id="Test" datafld="vorname">
</span><input type="button" value="Test"
onClick="alert(document.all.Test.recordNumber)">
</td>
</tr>
```

Das Beispiel stellt eine Tabellenzeile dar, in der ein Datenfeld ausgegeben wird (die übrigen Befehle zur Datenanbindung fehlen hier). Zu Testzwecken ist ein Button eingefügt, bei dessen Anklicken die Datensatznummer des aktuell angezeigten Datensatzes ausgegeben wird.



Lesen

**sourceIndex**

Speichert die Information, das wievielte HTML-Element ein Element innerhalb einer HTML-Datei ist. Das Beispiel enthält verschiedene Elemente, unter anderem einen Absatz mit dem id-Namen "Absatz". Außerdem enthält das Beispiel einen Verweis. Beim Anklicken des Verweises wird mit `alert()` ausgegeben, das wievielte Element der Absatz mit besagtem Namen innerhalb der HTML-Datei ist. Im Beispiel ist es das 7. Element, denn es werden alle einleitenden und allein stehenden HTML-Tags der gesamten HTML-Datei gezählt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p>Etwas Text mit<br>Zeilenumbruch</p>
<p id="Absatz">HTML-Element, aber das wie viele?</p>
<a href="javascript:alert(document.all.Absatz.sourceIndex)">Test</a>
</body></html>
```



Lesen

**tagName**

Speichert den HTML-Tagnamen eines Elements. Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird mit `alert()` der HTML-Tagname desjenigen Elements ausgegeben, das den id-Namen "DieseDatei" hat. Im Beispiel ist es das `<body>`-Tag. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body id="DieseDatei">
<a href="javascript:alert(document.all.DieseDatei.tagName)">Test</a>
</body></html>
```



Lesen

Ändern

**title**

Speichert den Wert, der im Attribut `title=` eines Elements vergeben wurde. Dieses Attribut ist beim MS Internet Explorer bei allen sichtbaren Elementen erlaubt und bewirkt beim Darüberfahren mit der Maus, dass eine Art Tooltip (Hilfetext-Popup) zu dem Element angezeigt wird. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p id="Absatz" title="Hilfe zum Text"
onClick="alert(document.all.Absatz.title)">Text</p>
</body></html>
```

Das Beispiel enthält einen Textabsatz. Dieser Textabsatz erhält mit dem Attribut `title=` einen Tooltip-Text. Beim Darüberfahren mit der Maus erscheint der Tooltip als kleines Popup-Fenster. Beim Anklicken des Absatzes wird der gleiche Text mit `alert()` ausgegeben.



### *click()*

Bewirkt, dass ein Element angeklickt wird, so wie wenn der Anwender mit der Maus auf das Element klickt. Erwartet keine Parameter. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a id="V1" href="javascript:alert('Verweis 1 geklickt')">Verweis 1</a><br>
<a href="javascript:document.all.V1.click()">Verweis 2</a><br>
</body></html>
```

Das Beispiel enthält zwei Verweise. Beim Klicken auf den ersten Verweis wird mit `alert()` die Meldung »Verweis 1 geklickt« ausgegeben. Beim Klicken auf den zweiten Verweis wird die Methode `click()` auf den ersten Verweis angewendet. Dadurch wird dieser scriptgesteuert angeklickt, und es wird trotz Klickens auf den zweiten Verweis die Meldung »Verweis 1 geklickt« ausgegeben.



### *contains()*

Ermittelt, ob ein Element ein bestimmtes anderes Element enthält. Erwartet als Parameter den `id`-Namen des gesuchten inneren Elements. Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion überprüft, ob das `<body>`-Tag, das den `id`-Namen `dieseDatei` hat, ein Element enthält, das den `id`-Namen `Absatz` hat. Wenn ja, wird der innere Text dieses Elements mit `alert()` ausgegeben:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
  if(document.all.dieseDatei.contains(Absatz))
    alert(document.all.Absatz.innerText);
}
//-->
</script>
</head><body id="dieseDatei">
<p id="Absatz">Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>
```



### **getAttribute()**

Ermittelt, ob ein Element in seinem HTML-Tag ein bestimmtes Attribut enthält oder nicht. Gibt den Wert zurück, den das Attribut hat, falls das Attribut gefunden wird, ansonsten die leere Zeichenkette "". Erwartet folgende Parameter:

- *Attribut* = Name des gesuchten Attributs im HTML-Tag.
- *Groß-/Kleinschreibung* = `true` übergeben, wenn bei dem Attribut Groß-/Kleinschreibung unterschieden werden soll, oder `false`, wenn es egal ist, wie das Attribut geschrieben ist.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
  if(document.all.Absatz.getAttribute("align","false") == "center")
    document.all.Absatz.setAttribute("align","right","false");
}
//-->
</script>
</head><body>
<p id="Absatz" align=center>Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>
```

Das Beispiel enthält einen zentriert ausgerichteten Absatz mit dem id-Namen Absatz und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion ermittelt mit `getAttribute()`, ob das Element mit dem Namen Absatz ein Attribut `align` enthält. Außerdem wird abgefragt, ob der Wert dieser Angabe gleich `center` ist. Da dies im Beispiel der Fall ist, wird die abhängige Anweisung ausgeführt. Darin wird das Attribut `align=` mit Hilfe von `setAttribute()` dynamisch geändert, sodass der Absatz hinterher rechtsbündig ausgerichtet ist.



### **insertAdjacentHTML()**

Fügt zusätzlichen HTML-Code vor oder nach einem Element ein. Erwartet folgende Parameter:

- *Position* = Angabe, wo genau der HTML-Code hinzugefügt werden soll. Übergeben Sie `BeforeBegin`, wenn der Code vor dem einleitenden HTML-Tag des Elements eingefügt werden soll, `AfterBegin`, wenn der Code nach dem einleitenden Tag vor dem Inhalt eingefügt werden soll, `BeforeEnd`, wenn der Code am Ende des Inhalts vor dem abschließenden Tag eingefügt werden soll, oder `AfterEnd`, wenn der Code hinter dem abschließenden HTML-Tag eingefügt werden soll.
- *Code* = HTML-Code, der eingefügt werden soll.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
```

```

var i = 0;
var Nummerierung = "";
function Nummerieren() {
    Nummerierung = "<b>" + (i+1) + ".</b> ";
    if(i < document.all.tags("p").length)
        document.all.tags("p")[i].insertAdjacentHTML("AfterBegin",Nummerierung);
    i = i + 1;
}
//-->
</script>
</head><body>
<p>Ein Absatz</p>
<p>Ein anderer Absatz</p>
<p>Noch ein Absatz</p>
<a href="javascript:Nummerieren()">Nummerieren</a>
</body></html>

```

Das Beispiel enthält drei Absätze und einen Verweis. Bei jedem Klick auf den Verweis wird der jeweils nächste Absatz dynamisch durchnummeriert. Dazu wird beim Klick auf den Verweis die Funktion `Nummerieren()` aufgerufen. Diese Funktion verwaltet die beiden global definierten Variablen `i` (Zähler für die Nummerierung) und `Nummerierung` (HTML-formatierte Zeichenkette, die die Nummerierung hinzufügt). Solange der Zähler für die Nummerierung kleiner ist als die Anzahl der verfügbaren Absätze, wird im jeweils nächsten Absatz mit `insertAdjacentHTML()` hinter dem einleitenden `<p>`-Tag dynamisch der aktuelle Wert der Variablen `Nummerierung` eingefügt.



### **`insertAdjacentText()`**

Fügt zusätzlichen Text vor oder nach einem Element ein. Erwartet folgende Parameter:

- **Position** = Angabe, wo genau der Text hinzugefügt werden soll. Übergeben Sie `BeforeBegin`, wenn der Text vor dem einleitenden HTML-Tag des Elements eingefügt werden soll, `AfterBegin`, wenn der Text nach dem einleitenden Tag vor dem Inhalt eingefügt werden soll, `BeforeEnd`, wenn der Text am Ende des Inhalts vor dem abschließenden Tag eingefügt werden soll, oder `AfterEnd`, wenn der Text hinter dem abschließenden HTML-Tag eingefügt werden soll.
- **Text** = Text, der eingefügt werden soll.

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Ergaenzen() {
    document.all.DynText.insertAdjacentText("BeforeEnd"," wird dynamisch!");
}
//-->
</script>
</head><body>
<p id="DynText">HTML</p>
<a href="javascript:Ergaenzen()">Ergaenzen</a>
</body></html>

```

Das Beispiel enthält einen Absatz mit dem `id`-Namen `DynText`, der nur das Wort »HTML« enthält. Beim Anklicken des Verweises unterhalb wird die Funktion `Ergaenzen()` aufgerufen. Diese Funktion fügt vor dem abschließenden Tag den Text »wird dynamisch« ein.



### ***removeAttribute()***

Entfernt ein bestimmtes Attribut aus einem bestimmten HTML-Tag. Erwartet folgende Parameter:

- *Attribut* = Name des gesuchte Attributs im HTML-Tag.
- *Groß-/Kleinschreibung* = `true` übergeben, wenn bei dem Attribut Groß-/Kleinschreibung unterschieden werden soll, oder `false`, wenn es egal ist, wie das Attribut geschrieben ist.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
document.all.Absatz.removeAttribute("align","false");
}
//-->
</script>
</head><body>
<p id="Absatz" align=center>Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>
```

Das Beispiel enthält einen zentriert ausgerichteten Absatz mit dem `id`-Namen `Absatz` und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion entfernt aus dem Absatz das Attribut `align`, sodass der Absatz hinterher linksbündig ausgerichtet ist, weil das die Normaleinstellung für das `<p>`-Tag ist.

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.



### ***scrollIntoView()***

Positioniert den Inhalt im Anzeigefenster des Browsers so, dass ein bestimmtes Element angezeigt wird. Erwartet einen Parameter. Übergeben Sie `true`, wenn der Inhalt so positioniert werden soll, dass das Element oben im Anzeigefenster erscheint, oder `false`, wenn das Element unten im Anzeigefenster platziert sein soll. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
document.all.Absatz.scrollIntoView("true");
}
//-->
</script>
</head><body>
```

```

<p id="Absatz">Ein Text</p>
<script type="text/javascript">
<!--
for(i=1;i<100;i++)
  document.write("<br>Zeile " + i);
//-->
</script>
<p><a href="javascript:Test()">Test</a></p>
</body></html>

```

Das Beispiel enthält einen Textabsatz mit dem `id`-Namen `Absatz`. Dann schreibt das Beispiel in einem `Script`-Bereich mit der `write()`-Methode 100 Zeilen in die Datei, damit die Datei zu Testzwecken schön lang ist. Unterhalb davon ist dann ein Verweis notiert. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion positioniert den Inhalt des Anzeigefensters neu, und zwar so, dass der Textabsatz `Absatz` am oberen Fensterrand angezeigt wird.



### ***setAttribute()***

Fügt in einem bestimmten `HTML`-Tag ein bestimmtes Attribut hinzu. Erwartet folgende Parameter:

- *Attribut* = Name des gewünschten Attributs.
- *Wert* = die gewünschte Wertzuweisung für die Zusatzangabe.
- *Groß-/Kleinschreibung* = `true` übergeben, wenn bei dem Attribut Groß-/Kleinschreibung unterschieden werden soll, oder `false`, wenn es egal ist, wie das Attribut geschrieben wird.

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
document.all.Absatz.setAttribute("align","center","false");
}
//-->
</script>
</head><body>
<p id="Absatz">Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>

```

Das Beispiel enthält einen zentriert ausgerichteten Absatz mit dem `id`-Namen `Absatz` und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion fügt in dem Absatz das Attribut `align` hinzu, und zwar mit der Wertzuweisung `center`. Dadurch wird der Absatz hinterher zentriert ausgerichtet.

## 20.8 style

### Methoden

|                   |                               |
|-------------------|-------------------------------|
| getAttribute()    | (Stylesheet-Angabe ermitteln) |
| removeAttribute() | (Stylesheet-Angabe entfernen) |
| setAttribute()    | (Stylesheet-Angabe einfügen)  |

### style: Allgemeines zur Verwendung

Das Objekt `style` liegt in der JavaScript-Objekthierarchie nach dem Objektmodell des MS Internet Explorers ab Version 4.0 unterhalb des `all`-Objekts und regelt den Zugriff auf Stylesheet-Angaben, die für ein HTML-Element definiert sind. Sie können alle Sheet-Angaben auslesen und dynamisch ändern. So können Sie HTML-Elemente mit Hilfe von Scripts nach Belieben umformatieren.

Der Zugriff auf HTML-Elemente erfolgt dabei genau so wie beim `all`-Objekt (lesen Sie dazu beim `all`-Objekt vor allem die Passagen *HTML-Elemente ansprechen* und *HTML-Elemente mit Namen ansprechen*, Kapitel 23.7). Beim `style`-Objekt wird lediglich zusätzlich `style` notiert und dahinter die gewünschte Stylesheet-Angabe.

Auch im Document Object Model (DOM) der Version 2.0 gibt es das `style`-Objekt. Um auf die Eigenschaften und Methoden des `style`-Objekts nach DOM-Syntax zugreifen zu können, benötigen Sie einen Elementknoten. Um auf vorhandene Elementknoten im Dokument zuzugreifen, werden die Methoden des `document`-Objekts `getElementById`, `getElementsByName` und `getElementsByTagName` verwendet. Ausgehend davon können Sie angesprochene Elemente mit Hilfe von Scripts umformatieren.



### Beispiel für ältere Microsoft-Syntax (all):

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function MachGelb() {
  document.all.DynText.style.backgroundColor = "yellow";
  document.all.DynText.style.fontSize = "24pt";
  document.all.DynText.style.padding = "5mm";
}
//-->
</script>
</head><body>
<p id="DynText">Das ist der Text</p>
<a href="javascript:MachGelb()">gelb</a>
</body></html>
```

Das Beispiel enthält einen Textabsatz mit dem `id`-Namen `DynText`. Beim Anklicken des Verweises unterhalb wird die Funktion `MachGelb()` aufgerufen. Diese Funktion weist dem Textabsatz verschiedene Stylesheet-Angaben zu, zum Beispiel die Angabe für die Hintergrundfarbe (`backgroundColor`), und dabei den Wert für gelb (`yellow`). Der Textabsatz erhält dadurch beim Anklicken des Verweises dynamisch eine gelbe Hintergrundfarbe und die übrigen Attribute.

Es ist egal, ob das betreffende HTML-Element vorher bereits Stylesheet-Angaben enthält oder nicht. Im Beispiel enthält es keine.

Nach dem gleichen Schema können Sie alle CSS Stylesheet-Angaben behandeln, die der MS Internet Explorer kennt. Die Tabelle Style-Eigenschaften listet Stylesheet-Angaben auf. Beachten Sie dabei die Hinweise zur besonderen Schreibweise der Stylesheet-Angaben in JavaScript.



### Beispiel für DOM-Syntax:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function setCSS() {
  for(var i = 0; i < document.getElementsByTagName("p").length; i++) {
    document.getElementsByTagName("p")[i].style.border = "solid red 10px";
    document.getElementsByTagName("p")[i].style.backgroundColor = "#FF9933";
    document.getElementsByTagName("p")[i].style.color = "#FFFFFF";
    document.getElementsByTagName("p")[i].style.fontSize = "36pt";
  }
}
//-->
</script>
</head><body>
<p>Absatz</p>
<p>zweiter Absatz</p>
<p>dritter Absatz</p>
<p><a href="javascript:setCSS()">Stylesheet-Power!</a></p>
</body></html>
```

Das Beispiel enthält vier Textabsätze. Im letzten davon ist ein Verweis notiert, bei dessen Anklicken die Funktion `setCSS()` aufgerufen wird, die im Dateikopf notiert ist. Diese Funktion greift in einer `for`-Schleife der Reihe nach mit `document.getElementsByTagName("p")[i]` auf alle `p`-Elemente im Dokument zu. Mit `style` dahinter wird auf die CSS-Eigenschaften des Elements zugegriffen. Dahinter folgt die gewünschte CSS-Eigenschaft. Dieser wird jeweils ein gültiger Wert zugewiesen. Es ist egal, ob das betreffende HTML-Element vorher bereits Stylesheet-Angaben enthält oder nicht. Im Beispiel enthält es keine.

Nach dem gleichen Schema können Sie alle CSS Stylesheet-Angaben behandeln, die CSS 2.0 kennt. Die Tabelle zu Style-Eigenschaften listet im folgenden Abschnitt Stylesheet-Angaben auf. Beachten Sie dabei die Hinweise zur besonderen Schreibweise der Stylesheet-Angaben in JavaScript.

### Style-Eigenschaften

Eine wichtige Regel müssen Sie kennen: Wenn in einem Script eine CSS-Stylesheet-Angabe ausgelesen oder verändert werden soll, entfällt der Bindestrich, und der erste Buchstabe des Wortes hinter dem Bindestrich wird großgeschrieben. Die CSS-Stylesheet-Angabe `background-color` wird innerhalb eines JavaScripts also zu `backgroundColor`.



Die folgende Tabelle listet Style-Eigenschaften auf. Die Tabelle enthält links die Stylesheet-Angabe, wie Sie sie in JavaScript im Zusammenhang mit dem Style-Objekt notieren müssen. In der mittleren Spalte steht eine Kurzbeschreibung, was die Angabe bewirkt.

| <i>JavaScript-Angabe</i> | <i>Kurzbeschreibung</i>           |
|--------------------------|-----------------------------------|
| background               | Hintergrundbild                   |
| backgroundAttachment     | Wasserzeichen-Effekt              |
| backgroundColor          | Hintergrundfarbe                  |
| backgroundImage          | Hintergrundbild                   |
| backgroundPosition       | Position des Hintergrundbilds     |
| backgroundRepeat         | Wallpaper-Effekt                  |
| border                   | Rahmen                            |
| borderBottom             | Rahmen unten                      |
| borderBottomColor        | Rahmenfarbe unten                 |
| borderBottomStyle        | Rahmenart unten                   |
| borderBottomWidth        | Rahmendicke unten                 |
| borderColor              | Rahmenfarbe unten                 |
| borderLeft               | Rahmen links                      |
| borderLeftColor          | Rahmenfarbe links                 |
| borderLeftStyle          | Rahmenart links                   |
| borderLeftWidth          | Rahmendicke links                 |
| borderRight              | Rahmen rechts                     |
| borderRightColor         | Rahmenfarbe rechts                |
| borderRightStyle         | Rahmenart rechts                  |
| borderRightWidth         | Rahmendicke rechts                |
| borderStyle              | Rahmenart                         |
| borderTop                | Rahmen oben                       |
| borderTopColor           | Rahmenfarbe oben                  |
| borderTopStyle           | Rahmenart oben                    |
| borderTopWidth           | Rahmendicke oben                  |
| borderWidth              | Rahmendicke                       |
| bottom                   | Position von unten                |
| captionSide              | Tabellenbeschriftung              |
| clear                    | Fortsetzung bei Textumfluss       |
| clip                     | Anzeigebereich eingrenzen         |
| color                    | Textfarbe                         |
| cursor                   | Mauscursor                        |
| direction                | Schreibrichtung                   |
| display                  | Sichtbarkeit (ohne Platzhalter)   |
| emptyCells               | Darstellung leerer Tabellenzellen |
| float                    | Textumfluss                       |
| font                     | Schrift                           |
| fontFamily               | Schriftart                        |

| <i>JavaScript-Angabe</i> | <i>Kurzbeschreibung</i>                             |
|--------------------------|-----------------------------------------------------|
| fontSize                 | Schriftgröße                                        |
| fontStretch              | Schriftlaufweite                                    |
| fontStyle                | Schriftstil                                         |
| fontVariant              | Schriftvariante                                     |
| fontWeight               | Schriftgewicht                                      |
| height                   | Höhe eines Elements                                 |
| left                     | Position von links                                  |
| letterSpacing            | Zeichenabstand                                      |
| lineHeight               | Zeilenhöhe                                          |
| listStyle                | Listendarstellung                                   |
| listStyleImage           | Grafik für Aufzählungslisten                        |
| listStylePosition        | Listeneinrückung                                    |
| listStyleType            | Darstellungstyp der Liste                           |
| margin                   | Abstand/Rand                                        |
| marginBottom             | Abstand/Rand unten                                  |
| marginLeft               | Abstand/Rand links                                  |
| marginRight              | Abstand/Rand rechts                                 |
| marginTop                | Abstand/Rand oben                                   |
| maxHeight                | Maximalhöhe                                         |
| maxWidth                 | Maximalbreite                                       |
| minHeight                | Mindesthöhe                                         |
| minWidth                 | Mindestbreite                                       |
| overflow                 | übergroßer Inhalt- CSS-Beschreibung                 |
| padding                  | Innenabstand                                        |
| paddingBottom            | Innenabstand unten                                  |
| paddingLeft              | Innenabstand links                                  |
| paddingRight             | Innenabstand rechts                                 |
| paddingTop               | Innenabstand oben                                   |
| pageBreakAfter           | Seitenumbruch danach                                |
| pageBreakBefore          | Seitenumbruch davor                                 |
| position                 | Positionsart                                        |
| right                    | Position von rechts                                 |
| scrollbar3dLightColor    | Farbe für 3-D-Effekte (Scrollbars)                  |
| scrollbarArrowColor      | Farbe für Verschiebepfeile (Scrollbars)             |
| scrollbarBaseColor       | Basisfarbe der Scroll-Leiste (Scrollbars)           |
| scrollbarDarkshadowColor | Farbe für Schatten (Scrollbars)                     |
| scrollbarFaceColor       | Farbe für Oberfläche (Scrollbars)                   |
| scrollbarHighlightColor  | Farbe für oberen und linken Rand (Scrollbars)       |
| scrollbarShadowColor     | Farbe für unteren und rechten Rand (Scrollbars)     |
| scrollbarTrackColor      | Farbe für freibleibenden Verschiebeweg (Scrollbars) |
| tableLayout              | Tabellentyp                                         |
| textAlign                | Ausrichtung                                         |
| textDecoration           | Textdekoration                                      |
| textIndent               | Texteinrückung                                      |

| JavaScript-Angabe | Kurzbeschreibung                |
|-------------------|---------------------------------|
| textTransform     | Texttransformation              |
| top               | Position von oben               |
| verticalAlign     | vertikale Ausrichtung           |
| visibility        | Sichtbarkeit (mit Platzhalter)  |
| width             | Breite eines Elements           |
| wordSpacing       | Wortabstand                     |
| zIndex            | Schichtposition bei Überlappung |



### getAttribute()

Ermittelt, ob ein HTML-Element eine bestimmte Stylesheet-Angabe enthält bzw. welchen Wert. Gibt den Wert zurück, den die Stylesheet-Angabe hat, falls sie gefunden wird, ansonsten die leere Zeichenkette "". Erwartet folgende Parameter:

- *Style-Angabe* = die gesuchte Stylesheet-Angabe.
- *Groß-/Kleinschreibung* = `true` übergeben, wenn bei der Style-Angabe Groß-/Kleinschreibung unterschieden werden soll, oder `false`, wenn es egal ist, wie die Style-Angabe geschrieben ist.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
  var x
  if (document.all)
    x = document.all.Absatz.style.getAttribute("fontSize","false");
  else x=document.getElementById("Absatz").style.fontSize;
  alert(x);
}
//-->
</script>
</head><body>
<p id="Absatz" style="font-size:18pt">Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>
```

Das Beispiel enthält einen Absatz mit dem id-Namen Absatz und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion ermittelt mit `getAttribute()`, ob das Element mit dem Namen Absatz eine Stylesheet-Angabe `font-size` (CSS-Syntax) bzw. `fontSize` (JavaScript-Syntax) enthält. Da dies der Fall ist, wird der zugewiesene Wert (18pt) ermittelt. Dieser Wert wird in einem Meldungsfenster ausgegeben.

**Beachten Sie:** Wenn Sie nach DOM-Syntax programmieren, genügt es wie im `else`-Zweig des Beispiels, die CSS-Eigenschaft des obigen Absatzes mit folgender Anweisung abzufragen:

```
x = document.getElementById("Absatz").style.fontSize
```

Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.



### ***removeAttribute()***

Entfernt ein bestimmtes Attribut aus Stylesheet-Angaben. Erwartet folgende Parameter:

- *Attribut* = die gewünschte Stylesheet-Angabe im HTML-Tag.
- *Groß-/Kleinschreibung* = `true` übergeben, wenn bei der Stylesheet-Angabe Groß-/Kleinschreibung unterschieden werden soll, oder `false`, wenn es egal ist, wie die Stylesheet-Angabe geschrieben ist.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
  if (document.all)
    document.all.Absatz.style.removeAttribute("color","false");
  else document.getElementById("Absatz").style.color = "";
}
//-->
</script>
</head><body>
<p id="Absatz" style="color:red">Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>
```

Das Beispiel enthält einen Absatz mit dem id-Namen `Absatz` und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion entfernt aus dem Absatz die Stylesheet-Angabe `color`, sodass der Absatz hinterher die rote Farbe verliert und die Standardfarbe für Text erhält.

**Beachten Sie:** Wenn Sie nach DOM-Syntax programmieren, genügt es wie im `else`-Zweig des Beispiels, die CSS-Eigenschaft des obigen Absatzes mit folgender Anweisung zu entfernen:

```
document.getElementById("Absatz").style.color = ""
```

Durch Zuweisung einer leeren Zeichenkette wird die `Style`-Eigenschaft dynamisch entfernt. Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.



### ***setAttribute()***

Fügt in einem bestimmten HTML-Tag eine bestimmte Stylesheet-Angabe hinzu. Erwartet folgende Parameter:

- *Style-Angabe* = die gewünschte Stylesheet-Angabe.
- *Wert* = die gewünschte Wertzuweisung für die Stylesheet-Angabe.
- *Groß-/Kleinschreibung* = `true` übergeben, wenn bei der Stylesheet-Angabe Groß-/Kleinschreibung unterschieden werden soll, oder `false`, wenn es egal ist, wie die Stylesheet-Angabe geschrieben wird.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
if (document.all)
document.all.Absatz.style.setAttribute("border","thin solid red","false");
else document.getElementById("Absatz").style.border = "thin solid red";
}
//-->
</script>
</head><body>
<p id="Absatz">Ein Text</p>
<a href="javascript:Test()">Test</a>
</body></html>
```

Das Beispiel enthält einen Absatz mit dem id-Namen Absatz und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Test()` aufgerufen. Diese Funktion fügt in dem Absatz die Stylesheet-Angabe `border` hinzu, und zwar mit der Wertzuweisung `thin solid red`. Dadurch erhält der Absatz einen dünnen roten Rahmen.

**Beachten Sie:** Wenn Sie nach DOM-Syntax programmieren, genügt es wie im `else`-Zweig des Beispiels, die CSS-Eigenschaft des obigen Absatzes mit folgender Anweisung zu setzen:

```
document.getElementById("Absatz").style.border = "thin solid red"
```

Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.

## 20.9 anchors

### *Eigenschaft*

|        |                                          |
|--------|------------------------------------------|
| name   | (Name des Verweisankers)                 |
| length | (Anzahl der Verweisanker)                |
| text   | (Text des Verweisankers)                 |
| x      | (horizontale Position des Verweisankers) |
| y      | (vertikale Position des Verweisankers)   |

### *anchors: Allgemeines zur Verwendung*

Mit dem Objekt `anchors`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf Verweisanker, die in einer HTML-Datei definiert sind. Ein Verweisanker in HTML ist beispielsweise:

```
<a name="top">Hier beginnt die Seite</a>
```

In JavaScript 1.0 ist es lediglich möglich, die Anzahl der Verweisanker in einer Datei auszulesen. In JavaScript 1.2 sind jedoch weitere Eigenschaften implementiert worden. Es stehen folgende Arten zur Verfügung, mit JavaScript einen bestimmten Verweisanker anzusprechen:

## Schema 1 / Beispiel 1:

```
document.anchors[#].Eigenschaft  
document.anchors[0].name
```

## Schema 2 / Beispiel 2:

```
document.anchors["Ankername"].Eigenschaft  
document.anchors["oben"].text
```

Verweisanker können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1). Bei Verwendung von Indexnummern geben Sie `document.anchors` an und dahinter in eckigen Klammern, den wievielten Anker in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. den ersten Verweisanker sprechen Sie mit `anchors[0]` an, den zweiten Verweisanker mit `anchors[1]` usw. Beim Zählen gilt die Reihenfolge, in der die Verweisanker in der Datei notiert sind.
- Mit Namen des Ankers als Indexnamen (wie in Schema 2 / Beispiel 2). Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `document.anchors` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen, den Sie bei der Definition des Verweisankers im einleitenden `<a>`-Tag im Attribut `name=` angegeben haben.

**Beachten Sie:** Im MS Internet Explorer ist es nicht möglich, mit dem Schema 2 auf einen Verweisanker zuzugreifen. Sie können aber jeden Anker über das `all`-Objekt und dessen Eigenschaften ansprechen.



Speichert den Namen eines Verweisankers. Mit `document.anchors[0].name` können Sie den Namen eines Verweisankers in einer HTML-Datei ermitteln. Das Beispiel enthält zunächst HTML-Text mit einem Anker und einen Verweis. Beim Anklicken wird mit `alert()` der Name des Verweises ausgegeben.

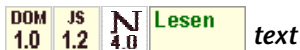
```
<html><head><title>Test</title>  
</head><body>  
<h1><a name="goethe">Meine Wahl</a></h1>  
<pre>Ich liebe mir den heiteren Mann  
Am meisten unter meinen Gästen:  
Wer sich nicht selbst zum besten haben kann,  
der ist gewiss nicht von den Besten.</pre>  
<p><a href="javascript:alert(document.anchors[0].name)">Der Ankername</a></p>  
</body></html>
```

**Beachten Sie:** Gegenwärtig ist es nur im Netscape 6.1 möglich, den Namen eines Ankers zu ändern.



Speichert die Information, wie viele Verweisanker eine HTML-Datei enthält. Mit `document.anchors.length` können Sie die Anzahl der Verweisanker in einer HTML-Datei ermitteln. Das Beispiel enthält zunächst HTML-Text mit einigen Ankern und schreibt am Ende mit `document.write()` dazu, wie viele Anker in der Datei gefunden wurden.

```
<html><head><title>Test</title>
</head><body>
<h1><a name="oben">Anfang</a></h1>
Am Anfang war, naja, seien wir ehrlich, wir wissen es nicht.
<h2><a name="mitte">Weisheit</a></h2>
Wem das Wasser bis zum Halse steht, der darf den Kopf nicht haengen lassen.
<h2><a name="unten">Ende</a></h2>
<script type="text/javascript">
<!--
  document.write("Gefunden wurden " + document.anchors.length + " Anker")
//-->
</script>
</body></html>
```



Speichert den Text eines Verweisankers. Mit `document.anchors['mitte'].text` können Sie den Text eines Verweisankers zwischen `<a>` und `</a>` ermitteln. Das Beispiel enthält zunächst HTML-Text mit einigen Ankern und einen Verweis. Beim Anklicken des Verweises wird mit `alert()` der Text des Ankers ausgegeben.

```
<html><head><title>Test</title>
</head><body>
<h1><a name="oben">Steuer</a></h1>
Der Mensch im gesellschaftlichen Leben kann sein Gut nicht genießen,
ohne es mit dem Staat zu teilen.
<h2><a name="mitte">Einsicht</a></h2>
Das Werk eines Philosophen ist viel leichter zu verbrennen, als zu widerlegen.
<h2><a name="unten">Ende</a></h2>
<p><a href="javascript:alert(document.anchors['mitte'].text)">mittlerer
Ankertext</a></p>

</body></html>
```



Mit `document.anchors[0].x` können Sie die horizontale Position eines Verweisankers in einer HTML-Datei ermitteln. Das Beispiel enthält einen Verweisanker und einen Verweis. Beim Anklicken wird die Position des Ankers vom linken Fensterrand aus gesehen ausgegeben.

```
<html><head><title>Test</title>
</head><body>
<a name="oben">ein Anker</a>
```

```
<p><a href="javascript:alert(document.anchors[0].x)">Position von links</a></p>
</body></html>
```



Mit `document.anchors[0].y` können Sie die vertikale Position eines Verweisankers in einer HTML-Datei ermitteln. Das Beispiel enthält einen Verweisanker und einen Verweis. Beim Anklicken wird die Position des Ankers vom oberen Fensterrand aus gesehen ausgegeben.

```
<html><head><title>Test</title>
</head><body>
<a name="oben">ein Anker</a>
<p><a href="javascript:alert(document.anchors[0].y)">Position von oben</a></p>
</body></html>
```

## 20.10 applets

### Eigenschaft

length (Anzahl Java-Applets)

### applets: Allgemeines zur Verwendung

Mit dem Objekt `applets`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf Java-Applets, die in einer HTML-Datei definiert sind. Die einzige gewöhnliche JavaScript-Objekteigenschaft des `applet`-Objekts ist die Anzahl der Java-Applets in einer Datei.

Über das `applets`-Objekt haben Sie jedoch auch Zugriff auf Code in Java-Applets. Dazu müssen Sie das gewünschte Java-Applet ansprechen. Es stehen drei Arten zur Verfügung, mit JavaScript ein bestimmtes Java-Applet anzusprechen:

Schema 1:

```
document.applets[#].Code();
```

Schema 2:

```
document.applets["AppletName"].Code();
```

Schema 3:

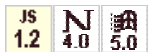
```
document.AppletName.Code();
```

Java-Applets können Sie auf drei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1): Bei Verwendung von Indexnummern geben Sie `document.applets` an und dahinter in eckigen Klammern, das wievielte Java-Applet in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. das erste Java-Applet sprechen Sie mit `applets[0]` an, das zweite Java-Applet mit `applets[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<applet>`-Befehle in der Datei notiert sind.



- Mit Namen des Java-Applets als Indexnamen (wie in Schema 2): Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `document.applets` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen, den Sie bei der Definition des Java-Applets im einleitenden `<applet>`-Tag im Attribut `name=` angegeben haben.
- Mit dem Namen des Java-Applets direkt (wie in Schema 3): Dabei geben Sie mit `document.AppletName` den Namen an, den Sie bei der Definition des Java-Applets im einleitenden `<applet>`-Tag im Attribut `name=` angegeben haben.



### Zugriff auf Code in Java-Applets

Sie können direkt auf den Code eines laufenden Java-Applets zugreifen. Dazu müssen Sie jedoch den Quellcode des Java-Applets kennen (Dateien `*.java` vor dem Kompilieren mit dem Java-Compiler).

Das folgende Beispiel zeigt das Prinzip. Nähere Informationen zur Vorgehensweise und zum Konzept des »lebendigen Drahts« (»LiveWire«) zwischen JavaScript und Java finden Sie auf den Entwicklerseiten von Netscape ([developer.netscape.com/](http://developer.netscape.com/)).

#### Beispiel Teil 1 – Quellcode eines Java-Applets:

```
import java.awt.Graphics;
import java.applet.Applet;

public class HelloWorld extends Applet
{
    String myString ="Das ist mein Hallo-Text";

    public void paint(Graphics g)
    {
        g.drawString(myString, 25, 20);
    }
    public void setString(String aString)
    {
        myString = aString;
        repaint();
    }
}
```

#### Beispiel Teil 2 – HTML-Datei mit Java-Applet und JavaScript-Zugriff:

```
<html><head><title>Test</title>
</head><body>
<applet code="HelloWorld" name="Hallo" width=150 height=25>
</applet><br>
<form name="Eingabe" action="">
<input name="Text">
<input type="button" value="Test"
onclick="document.Hallo.setString(document.Eingabe.Text.value)">
</form>
</body></html>
```

Im ersten Teil des Beispiels finden Sie den Quellcode eines Java-Applets, das einen Text am Bildschirm ausgibt. Auf den Quellcode wird hier nicht näher eingegangen. Achten Sie lediglich auf die letzte Methode, die in dem Quelltext definiert wird:

```
public void setString(String aString)
```

Damit wird der definierte Text letztendlich am Bildschirm ausgegeben. Im zweiten Teil des Beispiels wird angenommen, dass der Quellcode aus dem ersten Teil des Beispiels mit dem Java-Compiler zu einem Java-Applet mit dem Namen `HelloWorld.class` kompiliert wurde. Dieses Java-Applet wird in der HTML-Datei mit dem `<applet>`-Tag eingebunden. Dabei erhält das Applet mit `name=` den Namen "Hallo".

Unterhalb davon wird ein Formular mit einem Eingabefeld und einem Button definiert. In dem Eingabefeld lässt sich ein Text eingeben. Beim Klick auf den Button wird der eingegebene Text an das Java-Applet »gesendet«, und zwar so, dass der Text, den das Applet anzeigt, durch den eingegebenen Text ersetzt wird.

Dazu wird das Applet mit `document.Hallo` angesprochen. Dahinter wird keine JavaScript-Methode notiert, sondern direkt eine Methode aus dem Quellcode des Java-Applets, nämlich die Methode `setString()`. Diese Methode erwartet als Parameter eine Zeichenkette (siehe Quellcode des Java-Applets). Das JavaScript übergibt als Zeichenkette den Wert aus dem Eingabefeld des Formulars. Der Umgang mit Formularen in JavaScript wird beim `forms`-Objekt und seinen Unterobjekten näher beschrieben (Kapitel 20.11).

**Beachten Sie:** Opera 5.12 und der MS Internet Explorer 5.0 Macintosh Edition interpretierten dieses Beispiel nicht.



Speichert die Information, wie viele Java-Applets eine HTML-Datei enthält. Mit `document.applets.length` können Sie die Anzahl der Java-Applets in einer HTML-Datei ermitteln. Das Beispiel enthält zunächst HTML-Text mit einem Java-Applet und schreibt am Ende mit `document.write()` dazu, wie viele Applets in der Datei gefunden wurden.

```
<html><head><title>Test</title>
</head><body>
<applet code="HelloWorld.class" name="Hallo" width="150" height="25">
</applet>
<script type="text/javascript">
<!--
  document.write("Gefunden wurden " + document.applets.length + " Java-Applets")
//-->
</script>
</body></html>
```

## 20.11 forms

### Eigenschaften

`action` (Empfängeradresse bei Absenden)  
`encoding` (Kodierungstyp)

|        |                                 |
|--------|---------------------------------|
| length | (Anzahl Formulare in Datei)     |
| method | (Übertragungsmethode)           |
| name   | (Formularname)                  |
| target | (Zielfenster für CGI-Antworten) |

### Methoden

|               |                            |
|---------------|----------------------------|
| handleEvent() | (Ereignis verarbeiten)     |
| reset()       | (Formulareingaben löschen) |
| submit()      | (Formular abschicken)      |

### Unterobjekte

elements

### forms: Allgemeines zur Verwendung

Mit dem Objekt `forms`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf Formulare, die in einer HTML-Datei definiert sind. Es stehen folgende Arten zur Verfügung, mit JavaScript ein bestimmtes Formular anzusprechen:

Schema 1 / Beispiel 1:

```
document.forms[#].Eigenschaft
document.forms[#].Methode()

Ziel = document.forms[0].action;
document.forms[0].reset();
```

Schema 2 / Beispiel 2:

```
document.forms["FormularName"].Eigenschaft
document.forms["FormularName"].Methode

Ziel = document.forms["Testformular"].action;
document.forms["Testformular"].reset();
```

Schema 3 / Beispiel 3:

```
document.FormularName.Eigenschaft
document.FormularName.Methode()

Ziel = document.Testformular.action;
document.Testformular.reset();
```

Formulare können Sie auf drei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1): Bei Verwendung von Indexnummern geben Sie `document.forms` an und dahinter in eckigen Klammern, das wievielte Formular in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h., das erste Formular sprechen Sie mit `forms[0]` an, das zweite Formular mit `forms[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<form>`-Tags in der Datei notiert sind.

- Mit Namen des Formulars als Indexnamen (wie in Schema 2 / Beispiel 2): Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `document.forms` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen, den Sie bei der Definition des Formulars im einleitenden `<form>`-Tag im Attribut `name=` angegeben haben.
- Mit dem Namen des Formulars direkt (wie in Schema 3 / Beispiel 3): Dabei geben Sie mit `document.FormularName` den Namen an, den Sie bei der Definition des Formulars im einleitenden `<form>`-Tag im Attribut `name=` angegeben haben.

Die Eigenschaften und Methoden des `forms`-Objekts betreffen nur Bestandteile des gesamten Formulars, etwa die Kodiermethode. Um auf einzelne Eingabefelder, Buttons usw. zuzugreifen, steht das `elements`-Objekts mit seinen Unterobjekten zur Verfügung.



Speichert den Wert, der bei der Definition des Formulars im Attribut `action=` steht, also etwa eine `mailto`-Angabe oder ein Server-Programm, das die Formulardaten verarbeitet. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Bestaetigung() {
  var x = window.confirm("Dieses Formular geht an " + document.Testform.action);
  return x;
}
//-->
</script>
</head><body>
<form name="Testform" action="mailto:duselbst@deine.com"
onSubmit="return Bestaetigung()">
<input type="text" size="40" name="Eingabe">
<input type="submit" value="Absenden">
</form>
</body></html>
```

Das Beispiel enthält ein Formular. Wenn der Anwender das Formular durch Anklicken des `Submit`-Buttons abschickt, wird die Funktion `Bestaetigung()` aufgerufen, weil im einleitenden `<form>`-Tag der Event-Handler `onSubmit=` notiert ist. Dieser ermittelt den Rückgabewert der Funktion `Bestaetigung()`. Ist der Rückgabewert `true`, wird das Formular abgeschickt, ist er `false`, wird es nicht abgeschickt.

Innerhalb der Funktion `Bestaetigung()` erhält der Anwender mit der `confirm`-Methode ein Dialogfenster, in dem ihm der Wert der Formularangabe `action` angezeigt wird. Confirm-Fenster haben stets zwei Buttons, einen zum Bestätigen und einen zum Abbrechen. Der Button zum Bestätigen gibt bei Anklicken `true` zurück, der Abbrechen-Button `false`. Der Rückgabewert wird im Beispiel in der Variablen `x` gespeichert. Diese wird wiederum von der Funktion `Bestaetigung()` selbst zurückgegeben, denn der aufrufende Event-Handler erwartet ja einen solchen Rückgabewert.

**Beachten Sie:** Die Verwendung von `mailto`: als Zieladresse eines Formulars verursacht häufig Probleme. Mehr dazu im Kapitel 10.12.1 Formulare als E-Mail.

Sie können als Zieladresse eines Formulars auch eine HTML-Datei angeben und Formulardaten mittels JavaScript verarbeiten. Berücksichtigen Sie dabei, dass JavaScript nur dann Zugriff auf Daten hat, wenn die Methode `get` verwendet wurde. Opera 5.12 zeigt im Protokoll `file` den Quelltext der Zielformat an.



**encoding**

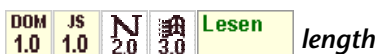
Speichert den Wert, der bei der Definition des Formulars im Attribut `encoding=` steht, also zum Beispiel »text/plain«. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CodeTyp() {
  if(document.Testform.action.indexOf("@") > 0)
    document.Testform.encoding = "text/plain";
  else
    document.Testform.encoding = "x-application/x-www-form-urlencoded";
  return true;
}
//-->
</script>
</head><body>
<form name="Testform" action="mailto:duselbst@deine.com"
onSubmit="return CodeTyp()">
<input type="text" size="40" name="Eingabe">
<input type="submit" value="Absenden">
</form>
</body></html>
```

Das Beispiel enthält ein Formular. Wenn der Anwender das Formular durch Anklicken das `Submit`-Buttons abschickt, wird die Funktion `CodeTyp()` aufgerufen. Innerhalb der Funktion `CodeTyp()` wird ermittelt, ob in der `action`-Angabe des Formulars das Zeichen `@` vorkommt, das eine E-Mail-Adresse verrät. Wenn ja, wird der Kodierungstyp auf `text/plain` gesetzt, ansonsten auf den für CGI-Programme üblichen Kodiertyp `x-application/x-www-form-urlencoded`. Die Funktion `CodeTyp()` gibt `true` zurück, denn der aufrufende Event-Handler `onSubmit` im einleitenden `<form>`-Tag erwartet einen solchen Rückgabewert. Das Formular wird nur abgeschickt, wenn der Rückgabewert `true` ist.

Um in einer Zeichenkette nach einem oder mehreren Zeichen zu suchen wie im Beispiel mit der Methode `indexOf()`, lesen Sie den Abschnitt über das `string`-Objekt.

**Beachten Sie:** Die Verwendung von `mailto:` als Zieladresse eines Formulars verursacht häufig Probleme. Mehr dazu im Kapitel 10.12.1 *Formulare als E-Mail*.



**length**

Speichert die Anzahl der Formulare, die in einer Datei definiert sind. Im Beispiel werden zwei kleine Formulare definiert. Am Ende wird mit Hilfe von `document.write()` zum Testen in die Datei geschrieben, wie viele Formulare darin definiert sind.

```

<html><head><title>Test</title>
</head><body>
<form name="Testform" action="mailto:duselbst@deine.com">
<input type="hidden" value="Guildo">
<input type="submit" value="Guildo">
</form>
<form name="Testform" action="mailto:duselbst@deine.com">
<input type="hidden" value="Heino">
<input type="submit" value="Heino">
</form>
<script type="text/javascript">
<!--
  document.write(document.forms.length + " Formulare");
//-->
</script>
</body></html>

```



Speichert den Wert, der bei der Definition des Formulars im Attribut `method=` steht, normalerweise »get« oder »post«. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Methode() {
  if(document.Testform.action.indexOf("@") > 0)
    document.Testform.method = "post";
  else
    document.Testform.method = "get";
  return true;
}
//-->
</script>
</head><body>
<form name="Testform" action="mailto:duselbst@deine.com"
onSubmit="return Methode()">
<input type="text" size="40" name="Eingabe">
<input type="submit" value="Absenden">
</form>
</body></html>

```

Das Beispiel enthält ein Formular. Wenn der Anwender das Formular durch Anklicken des `[Submit]`-Buttons abschickt, wird die Funktion `Methode()` aufgerufen. Innerhalb der Funktion `Methode()` wird ermittelt, ob in der `action`-Angabe des Formulars das Zeichen `@` vorkommt, das eine E-Mail-Adresse verrät. Wenn ja, wird die Versandmethode auf `post` gesetzt, ansonsten auf `get`. Die Funktion `Methode()` gibt `true` zurück, denn der aufrufende Event-Handler `onSubmit` im einleitenden `<form>`-Tag erwartet einen solchen Rückgabewert. Das Formular wird nur abgeschickt, wenn der Rückgabewert `true` ist. Um in einer Zeichenkette nach einem oder

mehreren Zeichen zu suchen wie im Beispiel mit der Methode `indexOf()`, lesen Sie den Abschnitt 20.32 über das `string`-Objekt.

**Beachten Sie:** Die Verwendung von `mailto:` als Zieladresse eines Formulars verursacht häufig Probleme. Lesen Sie hierzu das Kapitel 10.12.1 *Formulare als E-Mail*.



Speichert den Namen eines Formulars. Das Beispiel enthält ein Formular mit einem Eingabefeld. Unterhalb des Formulars steht ein Script, das in das Eingabefeld zur Kontrolle den Namen des Formulars schreibt. Im Beispiel ist das der Name `Testform`.

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="mailto:duselbst@deine.com">
<input type="text" size="40" name="Eingabe">
<input type="submit" value="Absenden">
</form>
<script type="text/javascript">
<!--
  document.Testform.Eingabe.value = document.Testform.name;
//-->
</script>
</body></html>
```



Speichert, welches Fenster im Falle eines Framesets nach dem Absenden aktiv werden soll. Das kann etwa bei Frame-Fenstern oder bei separaten Zweitfenstern interessant sein. Falls ein CGI-Programm HTML-Code an den Browser zurückschickt, zum Beispiel mit einem »Danke für das Absenden des Formulars«, dann erscheint die Ausgabe in dem Fenster mit dem angegebenen Namen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Ziel() {
  document.Testform.target = "unten";
  return true;
}
//-->
</script>
</head><body>
<form name="Testform" action="datei.htm"
onSubmit="return Ziel()">
<input type="text" size="40" name="Eingabe">
<input type="submit" value="Absenden">
</form>
</body></html>
```

Das Beispiel nimmt an, dass sich die Datei mit dem Formular in einem Frameset befindet, in dem es noch ein anderes Frame-Fenster gibt, und zwar eines mit dem Namen `unten`. Die Beispieldatei enthält ein Formular. Wenn der Anwender das Formular durch Anklicken des Submit-Buttons abschickt, wird die Funktion `Ziel()` aufgerufen. Innerhalb der Funktion `Ziel()` wird bestimmt, dass die HTML-Ausgabe das `action`-Ziels im Frame-Fenster `unten` erfolgt.

### **handleEvent()**

Übergibt ein Ereignis an ein Element, das in der Lage ist, auf das Ereignis zu reagieren. Funktioniert genauso wie `handleEvent()` beim `window`-Objekt (nähere Informationen siehe Kapitel 20.2).

### **reset()**

Löscht alle Eingaben in einem Formular und hat die gleiche Wirkung wie ein Button, der in HTML mit `type="reset"` definiert wurde. Das Beispiel enthält ein Formular mit zwei Eingabefeldern. Unterhalb des Formulars steht ein Verweis. Beim Anklicken des Verweises werden alle Eingaben in dem Formular gelöscht.

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
<input type="text" size="40" name="Feld1"><br>
<input type="text" size="40" name="Feld2"><br>
</form>
<a href="javascript:document.Testform.reset()">Formular leeren</a>
</body></html>
```

### **submit()**

Sendet ein Formular ab und hat die gleiche Wirkung wie ein Button, der in HTML mit `type="submit"` definiert wurde. Die JavaScript-Methode funktioniert seit Netscape 3.0 jedoch nur noch, wenn das Formular von einem Programm weiterverarbeitet wird, d.h. wenn im einleitenden `<form>`-Tag bei der Angabe `action=` zum Beispiel ein CGI-Programm aufgerufen wird. Wenn bei `action=` eine E-Mail-Adresse oder eine Newsgroup angegeben wird, funktioniert sie nicht. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="/cgi-bin/auswert.pl" method="get">
<input type="text" size="40" name="Feld1"><br>
<input type="text" size="40" name="Feld2"><br>
</form>
<script type="text/javascript">
<!--
function AbGehts()
{
    document.Testform.submit();
}
window.setTimeout("AbGehts()",60000);
```



```
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular mit Eingabefeldern. Unterhalb davon steht ein JavaScript. Darin wird mit der `setTimeout()`-Methode ein Countdown gestartet. Nach 60000 Millisekunden, also nach einer Minute, wird die Funktion `AbGehts()` aufgerufen. Diese schickt das Formular mit `submit()` ab.

## 20.12 elements

### Eigenschaften

|                             |                                  |
|-----------------------------|----------------------------------|
| <code>checked</code>        | (angekreuzt)                     |
| <code>defaultChecked</code> | (vorangekreuzt)                  |
| <code>defaultValue</code>   | (voreingegebener Wert)           |
| <code>form</code>           | (Name des zugehörigen Formulars) |
| <code>name</code>           | (Elementname)                    |
| <code>type</code>           | (Elementtyp)                     |
| <code>value</code>          | (Elementwert/-inhalt)            |

### Methoden

|                            |                             |
|----------------------------|-----------------------------|
| <code>blur()</code>        | (Element verlassen)         |
| <code>click()</code>       | (Element anklicken)         |
| <code>focus()</code>       | (auf Element positionieren) |
| <code>handleEvent()</code> | (Ereignis verarbeiten)      |
| <code>select()</code>      | (Text selektieren)          |

### Unterobjekte

`options`

### elements: Allgemeines zur Verwendung

Mit dem Objekt `elements`, das in der JavaScript-Objekthierarchie unterhalb des `forms`-Objekts liegt, haben Sie Zugriff auf Elemente eines Formulars. Es stehen folgende Arten zur Verfügung, mit JavaScript ein bestimmtes Formularelement anzusprechen:

Schema 1 / Beispiel 1:

```
document.forms[#].elements[#].Eigenschaft
document.forms[#].elements[#].Methode()

document.forms[0].elements[0].value = "Unsinn";
document.forms[0].elements[0].blur();
```

## Schema 2 / Beispiel 2:

```
document.FormularName.Elementname.Eigenschaft
document.FormularName.Elementname.Methode()

document.Testformular.Eingabe.value = "Unsinn";
document.Testformular.Eingabe.blur();
```

Formularelemente können Sie auf folgende Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1): Bei Verwendung von Indexnummern geben Sie `document.forms` an und dahinter in eckigen Klammern, das wievielte Formular in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h., das erste Formular sprechen Sie mit `forms[0]` an, das zweite Formular mit `forms[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<form>`-Tags in der Datei notiert sind. Dann folgt die Angabe `elements`. Auch dahinter notieren Sie wieder eine Indexnummer in eckigen Klammern. Auch dabei wird wieder bei 0 zu zählen begonnen, d.h. das erste Element innerhalb eines Formulars hat die Indexnummer 0, das zweite die Indexnummer 1 usw.
- Mit Namen (wie in Schema 2 / Beispiel 2): Dabei geben Sie mit `document.FormularName.ElementName` den Namen des Formulars und des Elements an, den Sie bei der Definition des Formulars und des Elements in den entsprechenden HTML-Tags im Attribut `name=` angegeben haben.

**Beachten Sie:** Hinter dem, was hier als `elements`-Objekt bezeichnet wird, verbergen sich in Wirklichkeit mehrere, allerdings sehr ähnliche JavaScript-Objekte. So gibt es Objekte für Eingabefelder, Auswahllisten, verschiedene Arten von Buttons. Alle diese Objekte werden jedoch im so genannten »elements-Array« zusammengefasst. Alle diese Objekte werden auf die gleiche Art und Weise angesprochen. Sie unterscheiden sich lediglich in ihren Eigenschaften und Methoden. Bei den Eigenschaften und Methoden in diesem Abschnitt ist daher immer vermerkt, für welche Objekte sie gelten. Folgende Angaben werden dabei unterschieden:

Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzeilige Eingabefelder, mehrzeilige Eingabefelder.

Bei den Eigenschaften und Methoden des `elements`-Objekts wird jeweils angegeben, auf welche dieser Elementtypen die Eigenschaft bzw. Methode anwendbar ist.

**Gruppenelemente:** Checkboxes und Radiobuttons bilden normalerweise Gruppen mehrerer zusammengehöriger Elemente, die eine gleiche Angabe zu `name=` haben und sich nur durch die Angabe `value=` unterscheiden. Solche Gruppen mit gleichen Namen bilden in JavaScript wiederum einen Array. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Formular" action="" method="get">
<input type="radio" name="Favoriten" value="Heino">
<input type="radio" name="Favoriten" value="Gildo">
<input type="radio" name="Favoriten" value="Marianne">
</form>
<script type="text/javascript">
<!--
```

```
document.Formular.Favoriten[2].checked = true;
//-->
</script>
</body></html>
```

Im Beispiel ist ein Formular mit mehreren Radiobuttons definiert. Die Gruppe zusammengehörender Radiobuttons hat den gleichen Namen. In JavaScript können Sie einzelne solcher Buttons ansprechen, indem Sie das Element (als Gruppe) ansprechen. Hinter dem Elementnamen notieren Sie dann in eckigen Klammern die Indexnummer des gewünschten Elements. Im Beispiel wird `Favoriten[2]` angesprochen. Das ist das dritte Element der Gruppe.

**Beachten Sie:** Auswahllisten kommen hier nicht vor. Um mit JavaScript auf solche Listen zuzugreifen, steht das Unterobjekt `options` zur Verfügung.



Speichert, ob bzw. dass ein Radio- oder Checkbutton aktiviert ist oder nicht. Mögliche Werte sind `true` bzw. `1` oder `false` bzw. `0`. Anwendbar auf: Klick-Buttons, Checkboxes, Radiobuttons. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Weiter() {
  if(document.Testform.Art[0].checked == true)
    window.location.href="frmdatei.htm"
  else if(document.Testform.Art[1].checked == true)
    window.location.href="datei.htm"
  else
    alert("Bitte eine Auswahl treffen");
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<input type="radio" name="Art" value="mit"> mit Frames
<input type="radio" name="Art" value="ohne"> ohne Frames
<br><input type="button" value="Starten" onClick="Weiter()">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit zwei Radiobuttons. Der Anwender kann dabei auswählen, ob er die folgenden Seiten des Projekts mit Frames oder ohne Frames sehen will. Beim Klick auf den Button `Starten` wird die Funktion `Weiter()` aufgerufen. Diese Funktion prüft, ob einer der beiden Radiobuttons aktiviert ist. Je nachdem, welcher von beiden aktiviert ist, wird eine unterschiedliche Datei aufgerufen (`location.href`). Wenn keiner der beiden Radiobuttons aktiviert ist, wird eine Fehlermeldung ausgegeben.

**Beachten Sie:** Sie können in Opera 5.12 unter Windows die Eigenschaft `checked` für Radioboxen erst dann ändern, wenn der Aufbau der Seite abgeschlossen ist.

**defaultChecked**

Speichert, ob bzw. dass ein Radio- oder Checkbutton per Voreinstellung aktiviert ist oder nicht. Mögliche Werte sind `true` bzw. `1` oder `false` bzw. `0`. Anwendbar auf: Checkboxes, Radio-buttons. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Weiter() {
  if(document.Testform.Art[0].checked == true)
    window.location.href="frminhalt.htm"
  else
    window.location.href="inhalt.htm"
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<input type="radio" name="Art" value="mit"> mit DHTML
<input type="radio" name="Art" value="ohne"> ohne DHTML
<br><input type="button" value="Starten" onClick="Weiter()">
</form>
<script language="JavaScript" type="text/javascript">
<!--
dhtml = false;
if(document.layers || document.all || document.getElementById) dhtml = true;
if(dhtml == true && document.Testform.Art[0].defaultChecked == false)
  document.Testform.Art[0].checked = true;
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular mit zwei Radiobuttons. Der Anwender kann dabei auswählen, ob er die folgenden Seiten des Projekts mit Dynamischem HTML oder ohne sehen will. Beim Klick auf den Button **Starten** wird die Funktion `Weiter()` aufgerufen. Diese Funktion prüft, ob einer der beiden Radiobuttons aktiviert ist. Je nachdem, welcher von beiden aktiviert ist, wird eine unterschiedliche Datei aufgerufen (`location.href`).

Unterhalb des Formulars ist jedoch noch ein weiterer Script-Bereich notiert. Der Script-Bereich steht unterhalb, weil der darin enthaltene Code sofort ausgeführt und die Existenz des Formulars aber bereits vorausgesetzt wird. Es wird abgefragt, ob eines der für Dynamisches HTML typischen Objekte `document.layers` oder `document.all` bzw. die typische Methode `document.getElementById` verfügbar sind. Wenn ja, wird eine Variable `dhtml` auf `true` gesetzt. In einer weiteren Abfrage wird mit `defaultChecked` überprüft, ob die Auswahlmöglichkeit, DHTML zu nutzen, im Formular als Vorauswahl aktiviert ist oder nicht. Wenn nicht, und wenn DHTML aber verfügbar ist (also wenn `dhtml` auf `true` gesetzt ist), wird die Auswahlmöglichkeit für DHTML nachträglich angekreuzt.

**Beachten Sie:** Opera 5.12 unter Windows interpretiert zwar die Eigenschaft `defaultChecked`, führt das obige Beispiel jedoch nicht aus, da die Änderung des Zustandes der Radiobox vor dem

Abschluss des Ladens der Seite erfolgt. Opera 5.02 kennt die Eigenschaft `defaultChecked` noch nicht.



### *defaultValue*

Speichert den voreingestellten Text eines Eingabefeldes. Anwendbar auf: einzeilige Eingabefelder, mehrzeilige Eingabefelder. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
URI: <input size="40" name="uri" value="http://www.xy.de/">
<input type="button" value="Gehe zu"
onClick="window.location.href=document.Testform.uri.value">
</form>
<script type="text/javascript">
<!--
if(navigator.userAgent.indexOf("en") > 0)
{
  document.Testform.uri.defaultValue = "http://www.xy.com/";
  document.Testform.uri.value = document.Testform.uri.defaultValue;
}
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular mit einem Eingabefeld und einem Button. Das Eingabefeld wird mit `http://www.xy.de/` vorgelegt. Beim Klick auf den Button wird der Wert aus dem Eingabefeld an `window.location.href` zugewiesen. Dadurch wird die entsprechende Adresse aufgerufen.

Bevor der Anwender jedoch überhaupt das Feld editieren kann, wird gleich beim Einlesen der Datei unterhalb des Formulars ein Script-Bereich notiert. Der Script-Bereich steht unterhalb, weil der darin enthaltene Code sofort ausgeführt und die Existenz des Formulars aber bereits vorausgesetzt wird. Es wird abgefragt, ob in der Ausweisung des Browsers (`navigator.userAgent`) der Bestandteil `en` vorkommt (`indexOf()`). Wenn ja, ist es ein englischsprachiger Browser. In diesem Fall wird der `defaultValue` des Eingabefeldes auf `http://www.xy.com/` geändert. Der geänderte Wert wird jedoch erst sichtbar, wenn er der Eigenschaft `value` zugewiesen wird. Dies geschieht in der zweiten Anweisung. Das Eingabefeld des Formulars ist dann abhängig von der Sprache des Browsers mit `http://www.xy.de/` oder `http://www.xy.com/` vorgelegt.

**Beachten Sie:** Opera 5.02 kennt die Eigenschaft `defaultValue` noch nicht. Trotzdem interpretiert auch dieser Browser das Beispiel. Das liegt daran, dass mit der Zuweisung `defaultValue="http://www.xy.com/"` der Wert der Eigenschaft manuell eingegeben wird. Sobald Sie jedoch ein JavaScript vom Originalwert abhängig machen möchten, müssen Sie berücksichtigen, dass diese Eigenschaft nicht existiert.



### *form*

Speichert das Formular, innerhalb dessen sich das Element befindet. Hinter der Eigenschaft `form` können Sie nochmals alle Eigenschaften und Methoden des `forms`-Objekts notieren. Anwendbar

auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzelne Eingabefelder, mehrzeilige Eingabefelder. Das Beispiel enthält ein Formular mit einem Button. Beim Anklicken des Buttons wird der Name des Formulars ausgegeben:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
<input type="button" name="derButton" value="Test"
onClick="alert(document.Testform.derButton.form.name)">
</form>
</body></html>
```



*name*

Speichert den Namen des Formularelements, wie er in HTML bei der Definition des Elements mit dem Attribut `name=` zugewiesen wurde. Anwendbar auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzelne Eingabefelder, mehrzeilige Eingabefelder. Das Beispiel enthält ein Formular mit einem Button. Beim Anklicken des Buttons wird der Name des Buttons ausgegeben:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
<input type="button" name="derButton" value="Test"
onClick="alert(document.Testform.derButton.name)">
</form>
</body></html>
```



*type*

Speichert den Elementtyp eines Formularelements. Anwendbar auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzelne Eingabefelder, mehrzeilige Eingabefelder. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
<input type="radio" name="Auto" value="Porsche"> Porsche<br>
<input type="radio" name="Auto" value="Mercedes"> Mercedes<br>
Name: <input size="30" name="Name"><br>
<input type="submit" name="Absendebuttton" value="Absenden">
</form>
<script type="text/javascript">
<!--
for(var i=0;i<document.Testform.length;++i)
{
  document.write("<br>Elementname: " + document.forms[0].elements[i].name);
  document.write(", Elementtyp: " + document.forms[0].elements[i].type);
}
```

```

}
//-->
</script>
</body></html>

```

Das Beispiel enthält ein Formular mit verschiedenen Elementen. Zu Testzwecken ist unterhalb des Formulars ein Script-Bereich notiert. Der Script-Bereich steht unterhalb, weil der darin enthaltene Code sofort ausgeführt und die Existenz des Formulars aber bereits vorausgesetzt wird. In dem Script wird für alle Elemente des Formulars mit `document.write()` der Elementname und der Elementtyp in die Datei geschrieben.

   **Lesen** **Ändern** **value**

Speichert den Wert, der in einem Formularelement eingegeben wurde bzw. diesem zugewiesen ist. Anwendbar auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzeilige Eingabefelder, mehrzeilige Eingabefelder. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
Name: <input size="30" name="Name">
<input type="submit" name="Absendebuttton" value="Absenden">
<input type="hidden" name="Browser" value="">
</form>
<script type="text/javascript">
<!--
document.Testform.Browser.value = navigator.userAgent;
document.write(document.Testform.Browser.value);
//-->
</script>
</body></html>

```

Das Beispiel enthält ein Formular, unter anderem mit einem versteckten Feld (`<input type="hidden">`). Unterhalb des Formulars ist ein Script-Bereich notiert. Der Script-Bereich steht unterhalb, weil der darin enthaltene Code sofort ausgeführt und die Existenz des Formulars aber bereits voraussetzt. In dem Script wird dem Wert des versteckten Formularfeldes (`document.Testform.Browser.value`) der Browser-Typ des Anwenders (`navigator.userAgent`) zugewiesen. Dieser dynamisch erzeugte Formularinhalt wird mit übertragen, wenn der Anwender das Formular abschickt. In obigen Beispiel wird zur Demonstration der Wert des versteckten Feldes mit der Methode `document.write()` in das Dokument geschrieben.

   **blur()**

Entfernt den Cursor bzw. den Fokus von dem betreffenden Element. Erwartet keine Parameter. Anwendbar auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzeilige Eingabefelder, mehrzeilige Eingabefelder. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>

```

```
<form name="Testform" action="">
<input name="Feld" size="30" onFocus="this.blur()">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einem Feld. Wenn der Anwender den Cursor in das Feld setzen will, wird der Cursor mit `this.blur()` sofort wieder aus dem Feld entfernt. Durch das Schlüsselwort `this` wird der Bezug zum aktuellen Objekt hergestellt, in diesem Fall das Element `Feld`, und die Methode auf das dieses Objekt angewendet. Eine solche Maßnahme kann beispielsweise sinnvoll sein, wenn Sie ein Feld ausschließlich als Ausgabefeld nutzen wollen, in dem der Anwender nichts editieren können soll.

**Beachten Sie:** Nicht immer wird der Cursor aus dem Eingabefeld entfernt. Eine Eingabe von Inhalt wird jedoch unterbunden. Opera 5.12 »kennt« zwar die Methode `blur()`, jedoch bleibt diese ohne Wirkung. Unter Opera 5.02 erzeugt die Anwendung der Methode einen JavaScript-Fehler.



Erzeugt einen automatischen Klick auf den betreffenden Button. Erwartet keine Parameter. Anwendbar auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, Abbrechen-Buttons, Absenden-Buttons. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
In welchem Jahr starb Goethe?<br>
<input size="6" name="Todesjahr">
<input type="button" name="derButton" value="Weiter" onClick="Check()">
</form>
<script type="text/javascript">
<!--
function Check() {
  if(document.Testform.Todesjahr.value != "1832")
    window.location.href = "nichts.htm";
  else
    window.location.href = "erfolg.htm";
}
window.setTimeout("document.Testform.derButton.click()",10000);
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular, in dem der Anwender das Todesjahr von Goethe eingeben soll. Wenn der Anwender nicht innerhalb von 10 Sekunden auf den Button `Weiter` klickt, geschieht dies von alleine. Dafür sorgt die letzte Anweisung in dem Script, das unterhalb des Formulars notiert ist. Unterhalb des Formulars ist ein Script-Bereich notiert. Der Script-Bereich steht unterhalb, weil der darin enthaltene Code sofort ausgeführt und die Existenz des Formulars aber bereits vorausgesetzt wird. Mit `document.Testform.derButton.click()` wird dort das Klicken des Buttons ohne Anwendereinwirkung bewerkstelligt. Der Aufruf der Methode ist in `window.setTimeout()` mit einer Verzögerungszeit von 10000 Millisekunden, also 10 Sekunden, eingebettet.



Beim Klick auf den Button, egal ob durch den Anwender oder automatisch, wird die Funktion `Check()` aufgerufen. Diese Funktion überprüft, ob in dem Eingabefeld für das Todesjahr der korrekte Wert steht. Abhängig davon wird eine andere Seite aufgerufen (`location.href`).

### **focus()**

Setzt den Cursor bzw. den Fokus auf das betreffende Element. Erwartet keine Parameter. Anwendbar auf: Klick-Buttons, Checkboxes, Felder für Datei-Upload, versteckte Elemente, Passwort-Felder, Radiobuttons, Abbrechen-Buttons, Absenden-Buttons, einzeilige Eingabefelder, mehrzeilige Eingabefelder. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Check() {
  if(document.Testform.Todesjahr.value != "1832")
  {
    alert("Sie wissen es einfach nicht!");
    document.Testform.Todesjahr.focus();
  }
  else
  {
    alert("Bravo!");
    window.location.href = "erfolg.htm";
  }
}
//-->
</script>
</head><body>
<form name="Testform" action="">
In welchem Jahr starb Goethe?<br>
<input size="6" name="Todesjahr">
<input type="button" value="Weiter" onClick="Check()">
</form>
</body></html>
```

Das Beispiel enthält ein Formular, in dem der Anwender das Todesjahr von Goethe eingeben soll. Wenn der Anwender auf den Button `Weiter` klickt, wird die Funktion `Check()` aufgerufen. Diese Funktion überprüft, ob in dem Eingabefeld für das Todesjahr der korrekte Wert steht. Abhängig davon wird entweder mit `alert()` eine bissige Meldung ausgegeben, oder es wird »Bravo« ausgegeben und eine andere Seite aufgerufen (`location.href`). Wenn der Anwender auf der Seite bleibt, weil die Eingabe fehlte oder falsch war, wird mit `document.Testform.Todesjahr.focus()` auf das Eingabefeld für das Todesjahr positioniert.

### **handleEvent()**

Übergibt ein Ereignis an ein Element, das in der Lage ist, auf das Ereignis zu reagieren. Funktioniert genau so wie `handleEvent()` beim `window`-Objekt (nähere Informationen siehe dort Kapitel 20.2).



Selektiert den gesamten in dem Feld enthaltenen Text. Erwartet keine Parameter. Anwendbar auf: Passwort-Felder, einzeilige Eingabefelder, mehrzeilige Eingabefelder. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
Kopieren Sie den folgenden Code in die Adresszeile Ihres Browsers:<br>
<input type="text" size="40" name="Code" value="javascript:top.close()">
</form>
<script type="text/javascript">
<!--
  document.Testform.Code.focus();
  document.Testform.Code.select();
//-->
</script>
</body></html>
```

Das Beispiel enthält ein Formular mit einem einzigen Feld. Das Feld wird mit dem Wert "javascript:top.close()" vorbelegt (wohlgemerkt: das ist keine JavaScript-Anweisung, sondern nur ein Feldinhalt). Unterhalb des Formulars ist ein Script-Bereich notiert. Der Script-Bereich steht unterhalb, weil der darin enthaltene Code sofort ausgeführt und die Existenz des Formulars aber bereits vorausgesetzt wird. Dort wird zunächst mit `document.Testform.Code.focus()` auf das Formularfeld positioniert. Denn das Selektieren von Text ist nur möglich, wenn das Feld den Fokus hat. Der vorbelegte Text des Formularfeldes wird dann mit `document.Testform.Code.select()` selektiert.

## 20.13 options

### Eigenschaften

|                              |                                   |
|------------------------------|-----------------------------------|
| <code>defaultSelected</code> | (voreingestellte Auswahl)         |
| <code>length</code>          | (Anzahl der Auswahlmöglichkeiten) |
| <code>selected</code>        | (aktuelle Auswahl)                |
| <code>selectedIndex</code>   | (Index der aktuellen Auswahl)     |
| <code>text</code>            | (Auswahltext)                     |
| <code>value</code>           | (Auswahlwert)                     |

### option: Allgemeines zur Verwendung

Mit dem Objekt `options`, das in der JavaScript-Objekthierarchie unterhalb des `elements`-Objekts liegt, haben Sie Zugriff auf Auswahllisten innerhalb eines Formulars. Sie können dabei auf jede einzelne Auswahlmöglichkeit der Auswahlliste zugreifen. Es stehen folgende Arten zur Verfügung, mit JavaScript auf die Optionen einer Auswahlliste zuzugreifen:

## Schema 1 / Beispiel 1:

```
document.forms[#].elements[#].options[#].Eigenschaft
document.forms[#].elements[#].Eigenschaft

document.forms[0].elements[0].options[4].text = "Unsinn";
document.forms[0].elements[0].selectedIndex = 2;
```

## Schema 2 / Beispiel 2:

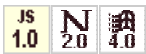
```
document.FormularName.Elementname.options[#].Eigenschaft
document.FormularName.Elementname.options.Eigenschaft

document.Testformular.Auswahl.options[4].text = "Unsinn";
document.Testformular.Auswahl.selectedIndex = 2;
```

Auswahllisten sind ganz normale Formularelemente. Auswahllisten sprechen Sie also an wie andere Formularelemente auch:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1): Bei Verwendung von Indexnummern geben Sie `document.forms` an und dahinter in eckigen Klammern, das wievielte Formular in der Datei Sie meinen. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. das erste Formular sprechen Sie mit `forms[0]` an, das zweite Formular mit `forms[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<form>`-Tags in der Datei notiert sind. Dann folgt die Angabe `elements`. Auch dahinter notieren Sie wieder eine Indexnummer in eckigen Klammern. Auch dabei wird wieder bei 0 zu zählen begonnen, d.h. das erste Element innerhalb eines Formulars hat die Indexnummer 0, das zweite die Indexnummer 1 usw. Sie müssen dabei die Indexnummer der gewünschten Auswahlliste ermitteln.
- Mit Namen (wie in Schema 2 / Beispiel 2): Dabei geben Sie mit `document.FormularName.ElementName` den Namen des Formulars und der Auswahlliste an, den Sie in HTML mit `<form name=>` und `<select name=>` vergeben.

Hinzu kommt bei Auswahllisten in einigen Fällen die Angabe `options` für einzelne Auswahlmöglichkeiten der Auswahlliste. Es gibt Objekteigenschaften von `options`, bei denen Sie diese Angabe, gefolgt von einer Indexnummer der gewünschten Auswahloption, angeben müssen. Die erste Auswahlmöglichkeit der Liste hat die Indexnummer 0, die zweite die Indexnummer 1 usw. Andere Objekteigenschaften benötigen keine Angabe von `options`. Bei den Beispielen auf dieser Seite wird jeweils gezeigt, wie Sie eine Eigenschaft des `options`-Objekts genau ansprechen.



### Neue Elemente in Auswahlliste einfügen

Sie können innerhalb eines JavaScripts neue Elemente zu einer Auswahlliste hinzufügen oder vorhandene Einträge durch neue ersetzen. Dazu müssen Sie mit Hilfe von JavaScript ein neues `option`-Objekt erzeugen und es einer Auswahlliste zuordnen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Hinzufuegen() {
  NeuerEintrag = new
Option(document.Testform.neu.value,document.Testform.neu.value,false,true);
```

```

document.Testform.Auswahl.options[document.Testform.Auswahl.length] =
NeuerEintrag;
document.Testform.neu.value = "";
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="8">
<option>Ein Eintrag</option>
</select>
<br>
<input type="text" name="neu">
<input type="button" value="Hinzu" onClick="Hinzufuegen()">
</form>
</body></html>

```

Im Beispiel wird ein Formular definiert, das eine Auswahlliste mit einem Eintrag, ein Eingabefeld und einen Button enthält. In dem Eingabefeld kann der Anwender Einträge eingeben, die er der Auswahlliste hinzufügen möchte. Beim Klick auf den Button wird der eingegebene Wert als Eintrag in die Auswahlliste übernommen.

Dazu wird die Funktion `Hinzufuegen()` aufgerufen, die im Dateikopf notiert ist. Mit der ersten Anweisung in dieser Funktion wird mit `new Option` ein neues option-Objekt erzeugt. Als Parameter wird unter anderem der Text übergeben, der bei dem neuen Listeneintrag angezeigt werden soll. Im Beispiel ist das der Wert des Eingabefeldes, das der Anwender ausgefüllt hat (`document.Testform.neu.value`). Anschließend müssen Sie angeben, an welcher Stelle in der Auswahlliste der neu erzeugte Eintrag eingefügt werden soll. Im obigen Beispiel soll der neue Eintrag jeweils ans Ende der Liste angehängt werden. Dazu wird der neue Eintrag einer Indexnummer der Auswahlliste zugeordnet, die mit der Eigenschaft `length` ermittelt wird. Diese Syntax können Sie stets benutzen, um Listeneinträge ans Ende einer Liste einzufügen. Doch wenn Sie, im obigen Beispiel den ersten Eintrag der Liste mit dem neuen Eintrag überschreiben wollten, dann müsste innerhalb der Funktion `Hinzufuegen()` dieser Befehl:

```
document.Testform.Auswahl.options[document.Testform.Auswahl.length] = NeuerEintrag;
```

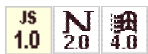
wie folgt lauten:

```
document.Testform.Auswahl.options[0] = NeuerEintrag;
```

Durch Angabe einer Indexnummer, die in der Liste schon vorkommt, überschreiben Sie also einen Eintrag. `new Option()` kennt vier Parameter, von denen die drei letzten Parameter optional sind.

- *text* = angezeigter Text in der Liste.
- *value* = zu übertragender Wert der Liste (optional).
- *defaultSelected* = `true` übergeben, wenn der Eintrag der defaultmäßig vorselektierte Eintrag sein soll, sonst `false` (optional).
- *selected* = `true` übergeben, wenn der Eintrag selektiert werden soll (optional). Opera 5.12 interpretiert die beiden letzten Parameter und Netscape 6.1 den letzten Parameter nicht.

**Beachten Sie:** Im Internet Explorer 5.0 können Sie nicht fensterübergreifend eine Auswahlliste ändern. Der Browser reagiert darauf mit einer Fehlermeldung bzw. mit einem Absturz. Notieren Sie deshalb die ändernde Funktion immer im gleichen Dokument wie die Liste und rufen Sie die Funktion gegebenenfalls fensterübergreifend auf.



### Elemente aus Auswahlliste löschen

Elemente können Sie löschen, indem Sie ihnen den Wert `null` zuordnen. Alternativ ist es auch möglich, dem Option-Array eine neue Länge zuzuweisen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Loeschen() {
  document.Testform.Auswahl.options[document.Testform.Auswahl.length-1] = null;
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="8">
<option>Auswahl 1</option>
<option>Auswahl 2</option>
<option>Auswahl 3</option>
<option>Auswahl 4</option>
<option>Auswahl 5</option>
</select>
<br>
<input type="button" value="Loeschen" onClick="Loeschen()">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einer Auswahlliste mit einem Button. Beim Anklicken des Buttons wird der jeweils letzte Eintrag aus der Auswahlliste gelöscht. Dazu wird die Funktion `Loeschen()` aufgerufen. Die erste Anweisung in der Funktion löscht einen Listeneintrag. Zum Löschen eines Eintrags geben Sie die Indexnummer des zu löschenden Eintrags an. Im Beispiel wird dabei `document.Testform.Auswahl.length-1` angegeben – das ist die Indexnummer des jeweils letzten Eintrags. Genauso gut können Sie aber auch 0 angeben, um den ersten Listeneintrag zu löschen, 1, um den zweiten zu löschen usw. Der Löschvorgang kommt zustande, indem Sie dem Listeneintrag den Wert `null` zuweisen.



### defaultSelected

Speichert, ob bzw. dass eine Auswahlmöglichkeit vorausgewählt ist. Mögliche Werte sind `true` oder `false`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function AuswahlReset() {
```

```

for(i=0; i<document.Testform.Auswahl.length; i++)
    if(document.Testform.Auswahl.options[i].defaultSelected == true)
        document.Testform.Auswahl.options[i].selected=true;
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="8">
<option>Augsburg</option>
<option>Berlin</option>
<option>Chemnitz</option>
<option>Dortmund</option>
<option selected>Essen</option>
<option>Frankfurt</option>
<option>Giessen</option>
<option>Hamburg</option>
</select>
<br>
<input type="button" value="Auswahl Reset" onClick="AuswahlReset()">
</form>
</body></html>

```

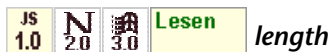
Das Beispiel enthält ein Formular mit einer Auswahlliste und einem Button. Bei der Auswahlliste wird bei der HTML-Definition ein Eintrag mit `selected` als Vorauswahl eingestellt. Der Anwender kann natürlich einen anderen Eintrag auswählen. Wenn er jedoch auf den Button mit der Aufschrift »Auswahl Reset« klickt, wird der Originalzustand wiederhergestellt. Dazu wird beim Anklicken des Buttons die Funktion `AuswahlReset()` aufgerufen. Diese Funktion ermittelt in einer `for`-Schleife für alle Auswahlmöglichkeiten der Auswahlliste, ob es sich um die Vorauswahl handelt:

```

(if(document.Testform.Auswahl.options[i].defaultSelected==true))

```

Denn die Vorauswahl ist zu diesem Zeitpunkt durchaus noch gespeichert, auch wenn der Anwender mittlerweile einen anderen Eintrag ausgewählt hat. Wenn die voreingestellte Auswahl gefunden ist, wird die aktuelle Auswahl auf den Indexwert der Vorauswahl gesetzt (`document.Testform.Auswahl.options[i].selected=true`). Dadurch wird der Originalzustand wiederhergestellt.



Speichert die Anzahl der Einträge einer Auswahlliste. Sie können die Anzahl der Einträge sowohl mit `document.Formularname.Listenname.length` als auch mit `document.Formularname.Listenname.options.length` bestimmen. Beide Angaben enthalten die gleichen Werte. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="8">
<option>Augsburg</option>

```

```

<option>Berlin</option>
<option>Chemnitz</option>
<option>Dortmund</option>
<option>Essen</option>
<option>Frankfurt</option>
<option>Giessen</option>
<option>Hamburg</option>
</select>
</form>
<script type="text/javascript">
<!--
document.write("Waehlen Sie einen der " + document.Testform.Auswahl.length + "
Eintraege aus");
//-->
</script>
</body></html>

```

Das Beispiel enthält ein Formular mit einer Auswahlliste. Unterhalb des Formulars ist ein Script-Bereich notiert. Unterhalb deshalb, weil die Anweisung in diesem Bereich sofort beim Einlesen der Datei ausgeführt wird und das Formular zu diesem Zeitpunkt bereits bekannt sein muss. Mit `document.write()` wird dynamisch die Anzahl der Auswahlmöglichkeiten der Auswahlliste in die Datei geschrieben.



Speichert, ob oder dass ein bestimmter Eintrag einer Auswahlliste selektiert ist. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CheckAuswahl() {
  if(document.Testform.Auswahl.options[2].selected == true)
  {
    alert("Diese Auswahl gilt nicht");
    return false;
  }
  return true;
}
//-->
</script>
</head><body>
<form name="Testform" onSubmit="return CheckAuswahl()" action="">
<select name="Auswahl" size="5">
<option>Goethe</option>
<option>Schiller</option>
<option>Gildo Horn</option>
<option>Homer</option>
<option>Fontane</option>
</select>
<br>

```

```
<input type="submit">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einer Auswahlliste und einem Submit-Button zum Absenden des Formulars. Beim Absenden wird jedoch über den Event-Handler `onSubmit=` im einleitenden `<form>`-Tag die Funktion `CheckAuswahl()` aufgerufen. Das Formular wird nur abgeschickt, wenn diese Funktion den Wert `true` zurückgibt. Die Funktion überprüft, ob der Anwender etwa auf die Idee gekommen ist, den Eintrag mit der Indexnummer 2 (Guido Horn) auszuwählen. In diesem Fall wird eine Meldung ausgegeben und es wird `false` zurückgegeben, wodurch das Formular nicht abgeschickt wird. Der Anwender kann eine neue Auswahl treffen.

**Beachten Sie:** Die Eigenschaft `selected` unterscheidet sich von `selectedIndex` durch die Adressierung des gewünschten Listeneintrags. Ansonsten haben beide Eigenschaften die gleiche Aufgabe.



Speichert, ob oder dass ein bestimmter Eintrag einer Auswahlliste selektiert ist. Wenn nichts ausgewählt ist, dann ist in der Eigenschaft der Wert `-1` gespeichert. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CheckAuswahl() {
  if(document.Testform.Auswahl.selectedIndex == 2)
    document.Testform.Auswahl.selectedIndex = 0;
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="5" onChange="CheckAuswahl()">
<option>Goethe</option>
<option>Schiller</option>
<option>Guido Horn</option>
<option>Homer</option>
<option>Fontane</option>
</select>
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einer Auswahlliste. Im einleitenden `<select>`-Tag ist der Event-Handler `onChange=` notiert. Dadurch wird jedes Mal, wenn der Anwender eine Auswahl in der Auswahlliste trifft, die Funktion `CheckAuswahl()` aufgerufen. Die Funktion überprüft, ob der Anwender etwa den Eintrag mit der Indexnummer 2 ausgewählt hat (Guido Horn). Wenn ja, wählt die Funktion selbstständig den Eintrag mit der Indexnummer 0 aus (Goethe). Das heißt, hinterher steht der Auswahlbalken auf diesem Eintrag. Der Anwender kann natürlich eine neue Auswahl treffen.



**Beachten Sie:** Die Eigenschaft `selectedIndex` unterscheidet sich von `selected` durch die Adressierung des gewünschten Listeneintrags. Ansonsten haben beide Eigenschaften die gleiche Aufgabe.



Speichert den angezeigten Text eines Eintrags in einer Auswahlliste. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CheckAuswahl() {
  if(document.Testform.Auswahl.selectedIndex == 2)
    document.Testform.Auswahl.options[2].text = "Thomas Mann";
}
//-->
</script>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="5" onChange="CheckAuswahl()">
<option>Goethe</option>
<option>Schiller</option>
<option>Guildo Horn</option>
<option>Homer</option>
<option>Fontane</option>
</select>
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einer Auswahlliste. Im einleitenden `<select>`-Tag ist der Event-Handler `onChange=` notiert. Dadurch wird jedes Mal, wenn der Anwender eine Auswahl in der Auswahlliste trifft, die Funktion `CheckAuswahl()` aufgerufen. Die Funktion überprüft, ob der Anwender etwa den Eintrag mit der Indexnummer 2 ausgewählt hat (Guildo Horn). Wenn ja, ändert die Funktion den angezeigten Text dieses Eintrags gemeinerweise auf »Thomas Mann«. Die Auswahlmöglichkeit bleibt selektiert.

**Beachten Sie:** Das Ändern der Objekteigenschaft `text` – so wie im obigen Beispiel – ist erst seit JavaScript 1.1 möglich (Netscape 3.x, MS Internet Explorer 4.x), das Auslesen dagegen wie ausgezeichnet seit JavaScript 1.0.



Speichert den internen Absendewert eines Eintrags in einer Auswahlliste. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CheckAuswahl() {
  for(i=0;i<document.Testform.Auswahl.length;++i)
    if(document.Testform.Auswahl.options[i].selected == true)
      alert(document.Testform.Auswahl.options[i].value);
}
```

```

}
//-->
</script>
</head><body>
<form name="Testform" action="">
<select name="Auswahl" size="5" onChange="CheckAuswahl()">
<option value="Idiot!">Blindauswahl 1</option>
<option value="Dummkopf!">Blindauswahl 2</option>
<option value="Bratze!">Blindauswahl 3</option>
<option value="Schlampe!">Blindauswahl 4</option>
<option value="Bingo - gewonnen!">Blindauswahl 5</option>
</select>
</form>
</body></html>

```

Das Beispiel enthält ein Formular mit einer Auswahlliste. In den einzelnen `<option>`-Tags sind mit `value=` interne Absendewerte notiert. Im einleitenden `<select>`-Tag ist außerdem der Event-Handler `onChange=` notiert. Dadurch wird jedes Mal, wenn der Anwender eine Auswahl in der Auswahlliste trifft, die Funktion `CheckAuswahl()` aufgerufen. Die Funktion ermittelt in einer `for`-Schleife für alle Auswahlmöglichkeiten der Auswahlliste, ob die jeweilige Auswahl selektiert wurde. Wenn ja, wird der interne Absendewert der Auswahlmöglichkeit in einem Meldungsfenster ausgegeben.

## 20.14 images

### Eigenschaften

|                       |                        |
|-----------------------|------------------------|
| <code>border</code>   | (Rahmen)               |
| <code>complete</code> | (Ladezustand)          |
| <code>height</code>   | (Höhe)                 |
| <code>hspace</code>   | (horizontaler Abstand) |
| <code>length</code>   | (Anzahl Grafiken)      |
| <code>lowsrc</code>   | (Vorschaugrafik)       |
| <code>name</code>     | (Name)                 |
| <code>src</code>      | (Grafikdatei)          |
| <code>vspace</code>   | (vertikaler Abstand)   |
| <code>width</code>    | (Breite)               |

### Methode

|                            |                        |
|----------------------------|------------------------|
| <code>handleEvent()</code> | (Ereignis verarbeiten) |
|----------------------------|------------------------|

### images: Allgemeines zur Verwendung

Mit dem Objekt `images`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf alle Grafiken, die in einer HTML-Datei definiert sind. Dabei können Sie auch vorhandene Grafiken dynamisch durch andere ersetzen. Ein neues Grafik-Objekt wird automatisch erzeugt, wenn der Web-Browser eine Grafik in der HTML-Datei vor-

findet. Es stehen folgende Arten zur Verfügung, mit JavaScript eine bestimmte Grafik anzusprechen:

#### Schema 1 / Beispiel 1:

```
document.images[#].Eigenschaft
document.images[#].Methode()

Hoehe = document.images[0].height;
```

#### Schema 2 / Beispiel 2:

```
document.BildName.Eigenschaft
document.BildName.Methode()

Hoehe = document.Portrait.height;
```

Grafikobjekte können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1): Bei Verwendung von Indexnummern geben Sie `document.images` an und dahinter in eckigen Klammern, die wievielte Grafik in der Datei Sie meinen. Jede Grafik, die in HTML mit dem `<img>`-Tag notiert wurde, zählt. Beachten Sie, dass der Zähler bei 0 beginnt, d.h., die erste Grafik sprechen Sie mit `images[0]` an, die zweite Grafik mit `images[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<img>`-Tags in der Datei notiert sind.
- Mit dem Namen der Grafik (wie in Schema 2 / Beispiel 2): Dabei geben Sie mit `document.BildName` den Namen an, den Sie bei der Definition der Grafik im einleitenden `<img>`-Tag im Attribut `name=` angegeben haben.

Für Grafiken, die Sie nachträglich mit JavaScript anzeigen möchten, müssen Sie jedoch eigene neue Grafikobjekte in JavaScript erzeugen. Das ist besonders dann wichtig, wenn Sie Grafiken dynamisch durch andere Grafiken ersetzen wollen. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
Zweitbild =new Image(104,102)
Zweitbild.src="cow2.gif";
document.write('');
alert("Weite: "+Zweitbild.width+"\nHoehe: "+Zweitbild.height);
//-->
</script>
</body></html>
```

Ein neues Grafik-Objekt speichern Sie in einem selbst vergebenen Objektnamen, im obigen Beispiel in dem Namen `Zweitbild`. Hinter dem Namen folgt ein Gleichheitszeichen. Dahinter folgt das reservierte Wort `new` und der Aufruf der Objektfunktion `Image()` (erster Buchstabe großgeschrieben!).

Anschließend können Sie mit der neu erzeugten Instanz arbeiten. So können Sie der Instanz über die Objekteigenschaft `src` eine Grafikdatei zuordnen, im Beispiel die Datei `blau.gif`. Die dritte Zeile im obigen Beispiel zeigt eine Anweisung, wie Sie dynamisch eine vorhandene Grafik

ersetzen können. Angenommen, in der HTML-Datei wurde als erste Grafik in der Datei mit `` ein rotes Bild eingebunden. Mit Hilfe von JavaScript können Sie nun, beispielsweise in Abhängigkeit vom Klicken auf einen Verweis, dieses rote Bild durch ein anderes, blaues Bild ersetzen.

Die Objektfunktion `Image()` kennt die zwei optionalen Parameter.

- *width* = Breite des Bildes
- *height* = Höhe des Bildes

Diese Parameter können Sie beim Erzeugen des `image`-Objektes mit angeben. Zur Demonstration wird im Beispiel die Höhe und die Breite der Grafik mit ausgegeben. Netscape 6.1 und Opera 5.12 interpretieren diese optionalen Parameter nicht und speichern jeweils den Wert 0 für die Höhe und Breite der Grafik.

**Beachten Sie:** Ein Anwendungsbeispiel für den Einsatz dynamisch austauschbarer Grafiken finden Sie in Kapitel 19.7, *Dynamische grafische Buttons*.



Speichert die Angabe zum Rahmen um eine Grafik, wie sie mit der Angabe `border=` im `<img>`-Tag möglich ist. Netscape speichert auch dann einen Wert, wenn die Angabe im HTML-Tag fehlt. Der MS Internet Explorer speichert nur dann einen Wert, wenn im HTML-Tag ein Wert angegeben ist. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:alert(document.Anna.border)">

</a>
</body></html>
```

Das Beispiel enthält eine Grafik, die in einen Verweis eingeschlossen ist. Beim Anklicken des Verweises wird mit `alert()` der Wert ausgegeben, der in der Angabe `border=` gespeichert ist.

**Beachten Sie:** Im Netscape 6.1 ist diese Angabe fehlerhaft und enthält als Rückgabewert `-1`. Das Ändern der Eigenschaft `border` ist bislang nur im MS Internet Explorer möglich. Im Netscape 6.1 wird der bestehende Rahmen bei einer Änderung entfernt.



Speichert die Information, ob eine Grafik vollständig geladen ist oder nicht. Enthält den Wert `true`, wenn die Grafik geladen ist, und den Wert `false`, wenn die Grafik nicht oder nicht vollständig geladen ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function BildCheck() {
  var Status = "";
  for(i = 0; i < document.images.length; ++i)
  {
```

```

if(document.images[i].complete == true)
    Status = Status + "Grafik " + (i+1) + " geladen\n";
else
    Status = Status + "Grafik " + (i+1) + " nicht geladen\n";
}
Status = Status + document.images.length + " Grafiken im Dokument";
alert(Status);
}
//-->
</script>
</head><body>
<br>
<br>
<br>
<a href="javascript:BildCheck()">Grafiken checken</a>
</body></html>

```

Das Beispiel enthält mehrere Grafiken und einen Verweis. Beim Anklicken des Verweises wird die Funktion `BildCheck()` aufgerufen, die im Dateikopf in einem Script-Bereich definiert ist. Diese Funktion ermittelt in einer `for`-Schleife für jede einzelne Grafik in der Datei (ermittelbar mit `document.images.length`), ob diese geladen ist (`if(document.images[i].complete == true)`) oder nicht. Das Ergebnis wird in einer Variablen `Status` gesammelt. Am Ende wird der Inhalt von `Status` mit `alert()` ausgegeben.

**Beachten Sie:** Die Eigenschaft `complete` wird von den Netscape-Browsern nicht mit den richtigen Werten belegt. Obwohl im vorliegenden Beispiel 2 Grafiken nicht geladen werden können, wird im Netscape 6.1 für alle Grafiken für die Eigenschaft `complete` der Wert `"true"` angegeben. Die älteren Netscape-Browser bis zur Version 4 dagegen geben für alle Grafiken in der Eigenschaft `complete` den Wert `"false"` an. Opera 5.12 und der MS Internet Explorer interpretieren diese Eigenschaft richtig.



Speichert die Angabe zur Höhe einer Grafik, wie sie mit der Angabe `height=` im `<img>`-Tag möglich ist. Es wird auch dann ein Wert gespeichert, wenn die Angabe im HTML-Tag fehlt. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>

<script type="text/javascript">
<!--
document.write(document.Christa.name + "<br>");
document.write(document.Christa.width + " x " + document.Christa.height + "
Pixel");
//-->
</script>
</body></html>

```

Das Beispiel enthält eine Grafik. Unterhalb davon ist ein Script-Bereich notiert. Darin wird mit `document.write()` der Bildname und die Angaben zu Breite und Höhe der Grafik geschrieben.

**Beachten Sie:** Im Netscape 6.1 wird in der Eigenschaft `height` der Wert 0 gespeichert, wenn das `height`-Attribut im HTML-Tag fehlt. Das Ändern der Eigenschaft `height` ist bislang nur im MS Internet Explorer und im Netscape 6.1 möglich.



### *hspace*

Speichert die Angabe zum horizontalen Abstand zwischen einer Grafik und ihren nebenliegenden Elementen, wie sie mit der Angabe `hspace=` im `<img>`-Tag möglich ist. Wenn die Angabe im HTML-Tag fehlt, hat die Objekteigenschaft `hspace` den Wert 0. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:alert(document.Dorian.hspace)">

</a>
</body></html>
```

Das Beispiel enthält eine Grafik, die in einen Verweis eingeschlossen ist. Beim Anklicken des Verweises wird mit `alert()` der Wert ausgegeben, der in der Angabe `hspace=` gespeichert ist.

**Beachten Sie:** Netscape 6.1 und Opera 5.12 interpretieren diese Angabe fehlerhaft. Im Netscape 6.1 ist stets -1 und in Opera 5.12 0 gespeichert. Das Ändern der Eigenschaft `hspace` ist bislang nur im MS Internet Explorer möglich. Im Netscape 6.1 wird der horizontale Abstand entfernt.



### *length*

Speichert die Anzahl der Grafiken in einer Datei. Das Beispiel enthält mehrere Grafiken. Am Ende steht ein Script-Bereich. Darin wird mit `document.write()` die Anzahl der Grafiken in die Datei geschrieben:

```
<html><head><title>Test</title>
</head><body>
<br>
<br>
<br>
<script type="text/javascript">
<!--
document.write("<hr>Seite enthält " + document.images.length + " Grafiken");
//-->
</script>
</body></html>
```



### *lowsrc*

Speichert die Angabe zur Vorab-Grafik, wie sie mit der Angabe `lowsrc=` im `<img>`-Tag möglich ist. Wenn die Angabe im HTML-Tag fehlt, enthält die Objekteigenschaft `lowsrc` keinen Wert. Wenn eine Angabe zu `lowsrc=` vorhanden ist, enthält die Objekteigenschaft bei Netscape den vollständigen URI, beim MS Internet Explorer, Netscape 6 und Opera 5.12 dagegen genau die Angabe, die in der Wertzuweisung an das HTML-Attribut steht.

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:alert(document.images[0].lowsrc)">

</a>
</body></html>
```

Das Beispiel enthält eine Grafik, die mit `lowsrc` eine vorab anzuzeigende Grafik definiert und in einen Verweis eingeschlossen ist. Beim Anklicken des Verweises wird mit `alert()` der Wert ausgegeben, der in der Angabe `lowsrc` gespeichert ist.



Speichert den Namen einer Grafik, wie er mit der Angabe `name` im `<img>`-Tag möglich ist. Wenn die Angabe im HTML-Tag fehlt, enthält die Objekteigenschaft `name` keinen Wert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>

<script type="text/javascript">
<!--
  document.write("<br>" + document.images[0].name);
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Grafik. Unterhalb davon ist ein Script-Bereich notiert. Darin wird mit `document.write()` der Bildname der Grafik in die Datei geschrieben.

**Beachten Sie:** Im Netscape 6.1 können Sie die Eigenschaft `name` auch ändern.



Speichert die Angabe, welche Grafikdatei angezeigt wird, wie sie mit der Angabe `src` im `<img>`-Tag möglich ist. Diese Eigenschaft lässt sich ändern. Dadurch können Sie Grafiken dynamisch durch andere ersetzen. Beachten Sie jedoch, dass Breite und Höhe ursprünglich in HTML-notierten Grafiken für alle dynamisch folgenden Grafiken übernommen werden. Um verzerrungsfreie Effekte zu erzielen, sollten also alle Grafiken, die einander dynamisch ersetzen, die gleiche Höhe und Breite haben. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>

<script type="text/javascript">
<!--
  var b = new Array();
  b[0] = new Image(); b[0].src = "holger.gif";
  b[1] = new Image(); b[1].src = "hilmar.gif";
  b[2] = new Image(); b[2].src = "heiner.gif";
```

```

b[3] = new Image(); b[3].src = "hans.gif";
var i = 0;
function Animation()
{
  if(i > 3) i = 0;
  document.images[0].src = b[i].src;
  i = i + 1;
  window.setTimeout("Animation()",1000);
}
window.setTimeout("Animation()",1000);
//-->
</script>
</body></html>

```

Das Beispiel enthält eine Grafik namens *ingo.gif*. Unterhalb davon ist ein Script-Bereich notiert, der dafür sorgt, dass die Grafik sich automatisch jede Sekunde in eine andere Grafik verwandelt. Dazu wird zunächst ein Array namens *b* notiert. Danach werden vier Elemente des Arrays *b* bestimmt. In allen vier Fällen handelt es sich um neue Grafikobjekte. Jedem der neuen Grafikobjekte wird mit der Eigenschaft *src* eine Grafikdatei zugewiesen. Beachten Sie, dass auch die Datei *hans.gif* wieder dabei ist, obwohl diese bereits angezeigt wird. Da sie in die Animation integriert werden soll, muss für sie aber trotzdem noch mal ein eigenes neues Grafikobjekt erzeugt werden.

Innerhalb der Funktion *Animation()*, die unterhalb der Funktion zum ersten Mal aufgerufen wird, wird das jeweils aktuelle Bild durch ein anderes ersetzt. Das geschieht durch Zuweisung von *b[i].src* an *document.images[0].src*. Da *b[i]* abhängig von der Variablen *i* ist, die bei jedem Funktionsaufruf verändert wird, wird jedes Mal ein anderes der vier Bilder angezeigt. Am Ende ruft sich die Funktion *Animation()* selbst wieder auf. So entsteht der Endlos-Effekt. Der Aufruf ist in einen *setTimeout()*-Aufruf eingebunden, der den nächsten Aufruf jeweils um 1000 Millisekunden, also um eine Sekunde, verzögert.

**Beachten Sie:** Im Internet Explorer und im Netscape 6 können Sie auch unterschiedlich große Grafiken verwenden. In diesem Fall sollten Sie die Angaben zur Größe der Grafik weglassen bzw. für die Höhe die Eigenschaft *height* und für die Weite die Eigenschaft *width* dynamisch ändern.



Speichert die Angabe zum vertikalen Abstand zwischen einer Grafik und den Elementen oberhalb und unterhalb davon, wie sie mit der Angabe *vspace=* im *<img>*-Tag möglich ist. Wenn die Angabe im HTML-Tag fehlt, hat die Objekteigenschaft *vspace* den Wert 0. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<a href="javascript:alert(document.Ina.vspace)">

</a>
</body></html>

```

Das Beispiel enthält eine Grafik, die in einen Verweis eingeschlossen ist. Beim Anklicken des Verweises wird mit *alert()* der Wert ausgegeben, der in der Angabe *vspace=* gespeichert ist.



**Beachten Sie:** Netscape 6.1 und Opera 5.12 interpretieren diese Angabe fehlerhaft. Im Netscape 6.1 ist stets -1 und in Opera 5.12 0 gespeichert. Das Ändern der Eigenschaft `vspace` ist bislang nur im MS Internet Explorer möglich. Im Netscape 6.1 wird der vertikale Abstand entfernt.



Speichert die Angabe zur Breite einer Grafik, wie sie mit der Angabe `width=` im `<img>`-Tag möglich ist. Es wird auch dann ein Wert gespeichert, wenn die Angabe im HTML-Tag fehlt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>

<script language="JavaScript" type="text/javascript">
<!--
document.write(document.Jochen.name + "<br>");
document.write(document.Jochen.width + " x " + document.Jochen.height + " Pixel");
//-->
</script>
</body></html>
```

Das Beispiel enthält eine Grafik. Unterhalb davon ist ein Script-Bereich notiert. Darin wird mit `document.write()` der Bildname und die Angaben zu Breite und Höhe der Grafik geschrieben.

**Beachten Sie:** Im Netscape 6.1 wird in der Eigenschaft `width` der Wert 0 gespeichert, wenn diese Angabe im HTML-Tag fehlt. Das Ändern der Eigenschaft `width` ist bislang nur im MS Internet Explorer und im Netscape 6.1 möglich.



Übergibt ein Ereignis an ein Element, das in der Lage ist, auf das Ereignis zu reagieren. Funktioniert genau so wie `handleEvent()` beim `window`-Objekt (nähere Informationen siehe Kapitel 20.2).

## 20.15 embeds

### Eigenschaften

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <code>height</code> | (Höhe des eingebetteten Objekts)                 |
| <code>hspace</code> | (horizontaler Abstand des eingebetteten Objekts) |
| <code>length</code> | (Anzahl eingebetteter Objekte)                   |
| <code>name</code>   | (Name des eingebetteten Objekts)                 |
| <code>src</code>    | (Quelle des eingebetteten Objekts)               |
| <code>width</code>  | (Breite des eingebetteten Objekts)               |
| <code>type</code>   | (Mime-Type des eingebetteten Objekts)            |
| <code>vspace</code> | (vertikaler Abstand des eingebetteten Objekts)   |

## Methoden

```
play()
stop()
```

### *embeds: Allgemeines zur Verwendung*

Mit dem Objekt `embeds`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf alle Multimedia-Elemente, die in einer HTML-Datei mit dem Netscape-Element `<embed>...</embed>` definiert sind. Dabei können Sie z.B. den Abspielvorgang von Sound-Dateien oder Videos dynamisch starten. Ein neues Objekt dieser Art wird automatisch erzeugt, wenn der Web-Browser eine Multimedia-Referenz nach Netscape-Syntax in der HTML-Datei vorfindet. Es stehen folgende Arten zur Verfügung, mit JavaScript eine bestimmte Multimedia-Referenz anzusprechen:

Schema 1 / Beispiel 1:

```
document.embeds[#].Eigenschaft
document.embeds[#].Methode()

Breite = document.embeds[0].width;
```

Schema 2 / Beispiel 2:

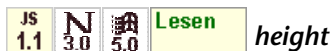
```
document.embeds["ObjektName"].Eigenschaft
document["ObjektName"].Methode()

Breite = document.embeds["Demovideo"].width;
```

Solche Objekte können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1): Bei Verwendung von Indexnummern geben Sie `document.embeds` an und dahinter in eckigen Klammern, das wievielte `embed`-Element in der Datei Sie meinen. Jedes Objekt, das in HTML mit dem `<embed>`-Tag notiert wurde, zählt. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. die erste Multimedia-Referenz sprechen Sie mit `embeds[0]` an, die zweite mit `embeds[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<embed>`-Tags in der Datei notiert sind.
- Mit dem Namen der Referenz (wie in Schema 2 / Beispiel 2): Dabei geben Sie mit `document.embeds["ObjektName"]` den Namen an, den Sie bei der Definition der Multimedia-Referenz im einleitenden `<embed>`-Tag im Attribut `name=` angegeben haben.

**Beachten Sie:** Ob Sie über das `embeds`-Objekt eine Multimedia-Referenz tatsächlich mit JavaScript ansprechen können, hängt zumindest bei Netscape davon ab, wie das Plug-In beschaffen ist, das diese Multimedia-Referenz anzeigt oder abspielt. Das Plug-In, also das Zusatzprogramm oder Programm-Modul, das etwa für das Abspielen eines Videos oder einer Sound-Datei im Browser zuständig ist, muss über eine Java-Schnittstelle verfügen. Andernfalls erhalten Sie bei Netscape JavaScript-Fehlermeldungen von der Sorte *Java object has no field or method named ....*



height

Speichert die Höhe eines eingebundenen Multimedia-Objekts. Das Beispiel enthält eine Multimedia-Referenz einer MIDI-Datei. Nachdem die HTML-Datei vollständig geladen ist, wird in

einem Meldungsfenster die Höhe des referenzierten Objekts ausgegeben. Dazu ist im `<body>`-Tag der Event-Handler `onLoad` notiert:

```
<html><head><title>Test</title></head>
<body onLoad="alert('Objekthöhe: ' + document.embeds[0].height + ' Pixel')">
<embed src="breeze.mid" width="300" height="200"></embed><br>
</body></html>
```

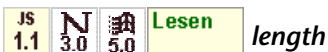


*hspace*

Speichert die Angabe zum horizontalen Abstand zwischen einer Multimedia-Referenz und ihren nebenliegenden Elementen, wie sie mit der Angabe `hspace=` im `<embed>`-Tag möglich ist. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<embed src="yippee.wav" width="200" height="100" hspace="30"
align="left"></embed><br>
<a href="javascript:alert(document.embeds[0].hspace)">hspace anzeigen!</a>
</body></html>
```

Das Beispiel enthält eine Multimedia-Referenz und einen Verweis. Beim Anklicken des Verweises wird mit `alert()` der Wert ausgegeben, der in der Angabe `hspace=` gespeichert ist.



*length*

Speichert die Anzahl der eingebetteten Objekte im Dokument. Das Beispiel enthält eine Multimedia-Referenz. Nach dem Laden des Dokumentes wird mit `alert()` die Anzahl der Multimedia-Elemente ausgegeben.

```
<html><head><title>Test</title></head>
<body onLoad="alert(document.embeds.length)">
<embed src="breeze.mid" width="300" height="200" type="audio/x-midi"></embed>
</body></html>
```



*name*

Speichert den Namen eines eingebetteten Objektes, wie er mit der Angabe `name=` im einleitenden `<embed>`-Tag vergeben worden ist. Wenn die Angabe im HTML-Tag fehlt, enthält die Objekteigenschaft `name` keinen Wert. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body>
<embed autostart="false" name="Musik" src="breeze.mid" width="300"
height="200"></embed>
<a href="javascript:document.embeds['Musik'].play()">Starten</a>
</body></html>
```

Das Beispiel enthält eine Multimedia-Referenz und einen Verweis. Nach Anklicken des Verweises wird mit `document.embeds['Musik']` das Multimedia-Element angesprochen und unter Verwendung der Methode `play()` der Abspielvorgang gestartet.

**src**

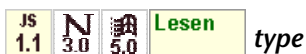
Datenquelle des Multimedia-Objektes. Im Beispiel ist eine Multimedia-Referenz definiert. Im nachfolgenden JavaScript-Bereich wird mit der Methode `link()` ein Verweis ins Dokument geschrieben. Der Methode wird dabei als Verweisziel die Quelldatei der Multimedia-Referenz durch Auslesen der `src`-Eigenschaft mit `document.embeds["Musik"].src` übergeben.

```
<html><head><title>Test</title></head>
<body>
<embed src="breeze.mid" autostart="true" name="Musik" width="300"
height="200"></embed> <br>
<script type="text/javascript">
<!--
  document.write("Download".link(document.embeds["Musik"].src));
//-->
</script>
</body></html>
```

**width**

Speichert die Breite eines eingebundenen Multimedia-Objekts. Das Beispiel enthält eine Multimedia-Referenz einer MIDI-Datei. Nachdem die HTML-Datei vollständig geladen ist, wird in einem Meldungsfenster die Breite des referenzierten Objekts ausgegeben. Dazu ist im `<body>`-Tag der Event-Handler `onLoad` notiert.

```
<html><head><title>Test</title></head>
<body onLoad="alert('Objektbreite: ' + document.embeds[0].width + ' Pixel')">
<embed src="breeze.mid" width="300" height="200"></embed><br>
</body></html>
```

**type**

Gibt den Mime-Type der Multimedia-Referenz an, so wie er im Attribut `type=` definiert wurde. Wurde diese Eigenschaft nicht definiert, so enthält die Eigenschaft `null` bzw. `undefined` im MS Internet Explorer. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body onLoad="alert('Objekttyp: ' + document.embeds[0].type)">
<embed src="breeze.mid" width="300" height="200" type="audio/x-midi"></embed><br>
</body></html>
```

Das Beispiel enthält eine Multimedia-Referenz einer MIDI-Datei. Nachdem die HTML-Datei vollständig geladen ist, wird in einem Meldungsfenster der Typ des referenzierten Objekts ausgegeben. Dazu ist im `<body>`-Tag der Event-Handler `onLoad` notiert.

**vspace**

Speichert die Angabe zum vertikalen Abstand zwischen einer Multimedia-Referenz und ihren darüber bzw. darunter liegenden Elementen, wie sie mit der Angabe `vspace=` im `<embed>`-Tag möglich ist. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
Eine Multimedia-Element<br>
<embed type="audio/x-wav" src="yippee.wav" width="200" height="100" vspace="30"
align="left"></embed><br>
<a href="javascript:alert(document.embeds[0].vspace)">Vertikalen Abstand
anzeigen!</a>
</body></html>
```

Das Beispiel enthält eine Multimedia-Referenz und einen Verweis. Beim Anklicken des Verweises wird mit `alert()` der Wert ausgegeben, der in der Angabe `vspace=` gespeichert ist.



Startet das Abspielen einer Multimedia-Referenz. Der Befehl spricht eine Schnittstelle des Abspielgerätes an. Seine Wirkungsweise ist deshalb immer abhängig vom verwendeten Plug-In und steht nicht in jedem Fall zur Verfügung. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body>
<embed type="audio/x-midi" autostart="false" src="breeze.mid" width="300"
height="200"></embed><br>
<a href="javascript:document.embeds[0].play()">Starten</a>
</body></html>
```

Das Beispiel enthält eine Multimedia-Referenz und einen Verweis. Nach Anklicken des Verweises wird mit `document.embeds[0].play()` das Multimedia-Element angesprochen und der Abspielvorgang gestartet.



Beendet das Abspielen einer Multimedia-Referenz. Der Befehl spricht eine Schnittstelle des Abspielgerätes an. Seine Wirkungsweise ist deshalb immer abhängig vom verwendeten Plug-In und steht nicht in jedem Fall zur Verfügung. Ein Beispiel:

```
<html><head><title>Test</title></head>
<body>
<embed type="audio/x-midi" autostart="true" src="breeze.mid" width="300"
height="200"></embed><br>
<a href="javascript:document.embeds[0].stop()">Stoppen</a>
</body></html>
```

Das Beispiel enthält eine Multimedia-Referenz und einen Verweis. Nach Anklicken des Verweises wird mit `document.embeds[0].stop()` das Multimedia-Element angesprochen und der Abspielvorgang beendet.

## 20.16 layers

### Eigenschaften

|              |                                             |
|--------------|---------------------------------------------|
| above        | (oberhalb liegender Layer)                  |
| background   | (Hintergrundbild eines Layers)              |
| bgColor      | (Hintergrundfarbe eines Layers)             |
| below        | (unterhalb liegender Layer)                 |
| clip         | (Anzeigebereich eines Layers)               |
| document     | (document-Objekt eines Layers)              |
| left         | (Links-Wert der linken oberen Ecke relativ) |
| length       | (Anzahl Layer)                              |
| name         | (Name eines Layers)                         |
| pageX        | (Links-Wert der linken oberen Ecke absolut) |
| pageY        | (Oben-Wert der linken oberen Ecke absolut)  |
| parentLayer  | (Objekt des Eltern-Layers)                  |
| siblingAbove | (Objekt des oberhalb liegenden Layers)      |
| siblingBelow | (Objekt des unterhalb liegenden Layers)     |
| src          | (externe Datei eines Layers)                |
| top          | (Oben-Wert der linken oberen Ecke relativ)  |
| visibility   | (Sichtbarkeit eines Layers)                 |
| zIndex       | (Schichtposition eines Layers)              |

### Methoden

|                  |                                             |
|------------------|---------------------------------------------|
| captureEvents()  | (Ereignisse überwachen)                     |
| handleEvent()    | (Ereignisse behandeln)                      |
| load()           | (externe Datei laden)                       |
| moveAbove()      | (über einen anderen Layer legen)            |
| moveBelow()      | (unter einen anderen Layer legen)           |
| moveBy()         | (Bewegen um Anzahl Pixel)                   |
| moveTo()         | (Bewegen zu Position relativ)               |
| moveToAbsolute() | (Bewegen zu Position absolut)               |
| releaseEvents()  | (Ereignisüberwachung beenden)               |
| resizeBy()       | (Breite und Höhe verändern um Anzahl Pixel) |
| resizeTo()       | (Breite und Höhe auf Anzahl Pixel setzen)   |
| routeEvent()     | (Event-Handler-Hierarchie durchlaufen)      |

### layers: Allgemeines zur Verwendung

Mit dem Objekt `layers`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf alle Layer, die in einer HTML-Datei definiert sind. Dieses Objekt ist – ebenso wie die entsprechenden HTML-Tags – Netscape 4-spezifisch. Es ist die Grundlage für Dynamisches Positionieren bei Netscape 4.x (in Kapitel 21.4 finden Sie auch weitere Beispiele).

**Wichtig:** Das `layers`-Objekt wird von Netscape ab Version 6.0 **nicht mehr** unterstützt. Es sollte also nur noch aus Gründen der Rückwärtskompatibilität zu Netscape 4.x Verwendung finden.

### Mit dem layers-Objekt arbeiten

Ein neues Layer-Objekt wird automatisch erzeugt, wenn der Web-Browser einen Layer in der HTML-Datei vorfindet. Es stehen folgende Arten zur Verfügung, mit JavaScript einen bestimmten Layer anzusprechen:

Schema 1 / Beispiel 1:

```
document.layers[#].Eigenschaft
document.layers[#].Methode()

Farbe = document.layers[0].bgColor;
```

Schema 2 / Beispiel 2:

```
document.LayerName.Eigenschaft
document.LayerName.Methode()

Farbe = document.Kopfbereich.bgColor;
```

Layer können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1 / Beispiel 1): Bei Verwendung von Indexnummern geben Sie `document.layers` an und dahinter in eckigen Klammern, den wievielten Layer in der Datei Sie meinen. Jeder Layer, der in HTML mit dem `<layer>`-Tag oder mit dem `<ilayer>`-Tag notiert wurde, zählt. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. den ersten Layer sprechen Sie mit `layers[0]` an, den zweiten mit `layers[1]` usw. Beim Zählen gilt die Reihenfolge, in der die `<layer>`/`<ilayer>`-Tags in der Datei notiert sind.
- Mit dem Namen des Layers (wie in Schema 2 / Beispiel 2): Dabei geben Sie mit `document.LayerName` den Namen an, den Sie bei der Definition der Grafik im einleitenden `<layer>`/`<ilayer>`-Tag im Attribut `name=` angegeben haben.

**JS 1.2** **N 4.0** **Lesen** *above*

Speichert die Information, welcher andere Layer über einem Layer liegt. Die Eigenschaft `above` liefert das Layer-Objekt des darüber liegenden Layers zurück. Sie müssen also noch eine Eigenschaft wie `name` anhängen, um mehr über den Layer zu erfahren. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<layer name="GelberLayer" top="100" left="50" height="200" bgcolor="#FFFFE0">
Inhalt des gelben Layers
</layer>
<layer top="150" name="LilaLayer" left="100" height="200" bgcolor="#FFE0FF">
Inhalt des lila Layers
</layer>
<a href="javascript:alert(document.GelberLayer.above.name)">
Wer liegt ueber dem gelben Layer?</a><br>
<a href="javascript:alert(document.GelberLayer.above.top)">
Und wo beginnt er?</a><br>
</body></html>
```

Im Beispiel werden zwei Layer mit den Namen `GelberLayer` und `LilaLayer` definiert. Außerdem sind zwei Verweise notiert. Der eine Verweis meldet beim Anklicken den Namen des Layers, der über dem gelben Layer liegt (`document.GelberLayer.above.name`), der andere Verweis dessen Position von oben (`document.GelberLayer.above.top`). Über dem gelben Layer liegt der lila Layer, da dieser in der Datei nach dem gelben Layer definiert wird.

### Lesen Ändern **background**

Speichert das Hintergrundbild (Wallpaper) eines Layers. Das folgende Beispiel zeigt, wie Sie ein solches Hintergrundbild dynamisch ändern können. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Neubild = new Image();
Neubild.src = "back02.gif";
function Hintergrund() {
  document.TestLayer.background.src="back02.gif";
}
//-->
</script>
</head><body>
<layer background="back03.gif" top="150" name="TestLayer"
left="100" height="200" bgcolor="#FF00FF">
Inhalt des Layers
</layer>
<a href="javascript:Hintergrund()">Anderer Hintergrund</a>
</body></html>
```

Im Beispiel wird ein Layer mit einem Hintergrundbild definiert. Beim Klick auf den Verweis wird das Hintergrundbild dynamisch durch ein anderes ersetzt. Beachten Sie, dass dazu in dem Script-Bereich im Dateikopf bereits beim Einlesen der Datei mit Hilfe des `images`-Objekts die Hintergrundgrafik registriert wird, die die andere dynamisch ersetzen soll.

Beim Klick auf den Verweis wird die Funktion `Hintergrund()` aufgerufen, die das Hintergrundbild dynamisch ändert. Hinter der Adressierung des Layers müssen Sie dabei `background.src` notieren und diesem Ausdruck die zuvor registrierte Grafik zuweisen.

### Lesen Ändern **bgColor**

Speichert die Hintergrundfarbe eines Layers. Im Beispiel wird ein Layer definiert. Beim Klick auf den Verweis wird die Funktion `Hintergrund()` aufgerufen, die im Dateikopf in einem Script-Bereich notiert ist. Diese Funktion fragt ab, welche Hintergrundfarbe der Layer gerade hat, und weist dementsprechend eine andere zu.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Farbe="aqua";
function Hintergrund() {
  if(document.TestLayer.bgColorFarbe=="aqua")
```



```

{ document.TestLayer.bgColor="yellow"; return; }
else
{ document.TestLayer.bgColor="aqua"; return; }
}
//-->
</script>
</head><body>
<layer top="150" name="TestLayer" left="100" height="200" bgcolor="aqua">
Inhalt des Layers
</layer>
<a href="javascript:Hintergrund()">Anderer Hintergrund</a>
</body></html>

```

**Beachten Sie:** Sie können die Hintergrundfarbe eines Layers nicht abfragen, da Netscape 4 merkwürdige `bgColor`-Werte zurückliefert, die nicht in das von Netscape dokumentierte Schema (hexadezimale Angabe oder Farbname) passen.



*below*

Speichert die Information, welcher andere Layer unter einem Layer liegt. Die Eigenschaft `below` liefert das Layer-Objekt des darunter liegenden Layers zurück. Sie müssen also noch eine Eigenschaft wie `name` anhängen, um mehr über den Layer zu erfahren. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<layer name="GelberLayer" top="100" left="50" height="200" bgcolor="#FFFFE0">
Inhalt des gelben Layers
</layer>
<layer top="150" name="LilaLayer" left="100"
height="200" bgcolor="#FFEE0F">
Inhalt des lila Layers
</layer>
<a href="javascript:alert(document.LilaLayer.below.name)">Wer liegt unter dem lila
Layer?</a><br>
<a href="javascript:alert(document.LilaLayer.below.left)">Und wo beginnt
er?</a><br>
</body></html>

```

Im Beispiel werden zwei Layer mit den Namen `GelberLayer` und `LilaLayer` definiert. Außerdem sind zwei Verweise notiert. Der eine Verweis meldet beim Anklicken den Namen des Layers, der unter dem lila Layer liegt (`document.LilaLayer.below.name`), der andere Verweis dessen Position von links (`document.GelberLayer.below.left`). Unter dem lila Layer liegt der gelbe Layer, da in der Datei zuerst der gelbe und danach der lila Layer definiert wird.



*clip*

Speichert die Information, welcher Ausschnitt eines Layers angezeigt wird. Dabei gibt es folgende Untereigenschaften:

- `clip.top` speichert den Pixelwert oben der linken oberen Ecke des sichtbaren Layer-Bereichs,

- `clip.left` speichert den Pixelwert links der linken oberen Ecke des sichtbaren Layer-Bereichs,
- `clip.bottom` speichert den Pixelwert unten der rechten unteren Ecke des sichtbaren Layer-Bereichs,
- `clip.right` speichert den Pixelwert rechts der rechten unteren Ecke des sichtbaren Layer-Bereichs,
- `clip.width` speichert die Breite des sichtbaren Layer-Bereichs (alternative Angabe zur Angabe einer rechten unteren Ecke),
- `clip.height` speichert die Höhe des sichtbaren Layer-Bereichs (alternative Angabe zur Angabe einer rechten unteren Ecke).

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  function MachKleiner()
  {
    with(document.MeinLayer)
    {
      if(clip.height > 0 && clip.width > 0)
        { clip.height-=10; clip.width-=10; }
    }
  }
  function MachGroesser()
  {
    with(document.MeinLayer)
    {
      if(clip.height < 200 && clip.width < 200)
        { clip.height+=10; clip.width+=10; }
    }
  }
//-->
</script>
</head><body>
<layer name="MeinLayer" top="80" left="80" height="200" width="200"
bgcolor="#FFFFE0">
Inhalt des Layers
</layer>
<a href="javascript:MachKleiner()">Kleiner</a><br>
<a href="javascript:MachGroesser()">Groesser</a>
</body></html>
```

Im Beispiel ist ein Layer definiert. Da innerhalb der HTML-Definition keine Angaben zum clip-Bereich des Layers gemacht werden, wird der gesamte Layer angezeigt. Seine `clip`-Werte ergeben sich aus der Größe des Layers. Intern sind also `clip.top` und `clip.left` gleich 0, und `clip.width`, `clip.height`, `clip.bottom` und `clip.right` haben alle den Wert 200, weil sich dies aus den Angaben `height=200`, `width=200` in der HTML-Definition des Layers ergibt.

Mit Hilfe von zwei Verweisen lässt sich im Beispiel der sichtbare Anzeigebereich des Layers verkleinern und wieder vergrößern. Die Verweise rufen die Funktionen `MachKleiner()` bzw. `MachGroesser()` auf, die in einem Script-Bereich im Dateikopf notiert sind. Die Funktion `MachKleiner()` fragt ab, ob `clip.width` und `clip.height` größer 0 sind. Wenn ja, werden beide Werte um 10 verringert, wodurch der Anzeigebereich des Layers von rechts unten um 10 Pixel beschnitten wird. Die Funktion `MachGroesser()` tut einfach das Umgekehrte.

### **document**

Jeder Layer wird von Netscape wie ein eigenes kleines Dokument behandelt. Deshalb gibt es für Layer auch ein `document`-Objekt, das genauso funktioniert wie das globale `document`-Objekt. Dadurch haben Sie Zugriff auf Dokument-Unterbjekte wie Grafiken, Formulare usw. Um solche Elemente innerhalb eines Layers anzusprechen, müssen Sie die Elemente über das Layer-eigene `document`-Objekt ansprechen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  function AndererInhalt()
  {
    document.MeinLayer.document.open();
    document.MeinLayer.document.write("Jetzt ist der Verweis weg!");
    document.MeinLayer.document.close();
  }
//-->
</script>
</head><body>
<layer name="MeinLayer" bgcolor="#FFFFE0">
<a href="javascript:AndererInhalt()">Diesen Verweis ueberschreiben</a>
</layer>
</body></html>
```

Im Beispiel wird ein Layer definiert, der einen Verweis enthält. Beim Anklicken wird die Funktion `AndererInhalt()` aufgerufen, die in einem Script-Bereich im Dateikopf notiert ist. Diese Funktion benutzt typische Methoden des `document`-Objekts wie `open()`, `write()` und `close()`. Das Besondere ist, dass es sich nicht um das globale Dokument handelt, sondern um das Dokument, das der Layer darstellt. Dies wird durch eine Adressierung wie `document.MeinLayer.document` erreicht.

**Beachten Sie:** Das Dokumentenmodell der Layer im Netscape hat auch Auswirkungen auf im HTML-Dokument definierte Formulare, Grafiken oder Verweise. Berücksichtigen Sie hier, dass alle Elemente, die sich in einem Layer befinden, zum Dokument des Layers gehören und nicht zum HTML-Dokument. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<layer name="MeinLayer" bgcolor="#FFFFE0" left="300">

</layer>
<script type="text/javascript">
<!--
```

```

    document.write("Anzahl der Bilder im HTML-Dokument: " + document.images.length);
    document.write("<br>Anzahl der Bilder im Layer: " +
        document.MeinLayer.document.images.length);
//-->
</script>
</body></html>

```

Im Beispiel wurde ein Layer definiert, der eine Grafik enthält. Weitere Bilder enthält die Datei nicht. Danach folgt ein JavaScript-Bereich, der mit der Methode `document.write` die Anzahl der Bilder im Dokument und im Layer ausgibt. Anhand des Beispiels sehen Sie, dass nur das Dokument des Layers eine Grafik enthält. Mit einer Adressierung wie `document.MeinLayer.document` können Sie diese Grafik ansprechen und z.B. deren Eigenschaft `src` ändern.



*left*

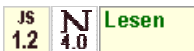
Speichert den Pixelwert für links von der linken oberen Ecke eines Layers. Bezug ist dabei das übergeordnete Dokument. Das kann entweder das globale Dokument sein oder ein Layer, innerhalb dessen der aktuelle Layer definiert ist. Für absoluten Bezug siehe `pageX`. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<layer name="SuperLayer" left="100">
  <layer name="Layerchen" left="200" width="300" bgcolor="#FFFFE0">
    <a href="javascript:alert(document.SuperLayer.document.Layerchen.left)">Links-
Wert?</a>
  </layer>
</layer>
</body></html>

```

Das Beispiel enthält einen Layer innerhalb eines anderen Layers, beginnend bei Pixelposition 200 von links, mit einem Verweis. Beim Anklicken des Verweises wird der Linkswert des inneren Layers ausgegeben. Der beträgt 200, obwohl der Layer optisch gesehen bei Position 300 beginnt, weil der übergeordnete Layer ja schon bei 100 beginnt und die 200 des inneren Layers dazu addiert werden.



*length*

Speichert die Anzahl der Layer in einer Datei. Das Beispiel enthält ein paar Layer zu Testzwecken und schreibt dann mit `document.write()` die Anzahl der definierten Layer in die Datei.

```

<html><head><title>Test</title>
</head><body>
<layer></layer>
<layer></layer>
<layer></layer>
<layer></layer>
<script type="text/javascript">
<!--
    document.write("Auf dieser Seite gibt es " + document.layers.length + " Layer");
//-->
</script>
</body></html>

```

JS 1.2 N 4.0 Lesen Ändern *name*

Speichert den Namen eines Layers. Das Beispiel enthält einen Layer mit einem Verweis. Beim Anklicken des Verweises wird der Name des Layers in einem Meldungsfenster ausgegeben. Es ist derjenige Name, der in HTML bei dem Attribut `name=` angegeben ist.

```
<html><head><title>Test</title>
</head><body>
<layer name="Layerchen" bgcolor="#FFFFE0">
<a href="javascript:alert(document.Layerchen.name)">Wie heiße ich denn?</a>
</layer>
</body></html>
```

JS 1.2 N 4.0 Lesen Ändern *pageX*

Speichert den Pixelwert für links von der linken oberen Ecke eines Layers. Bezug ist dabei das globale Dokument, auch wenn der aktuelle Layer innerhalb eines anderen Layers definiert ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  function NachLinks()
  {
    document.Layerchen.pageX = 0;
  }
//-->
</script>
</head><body>
<layer name="Layerchen" left="300" width="300" bgcolor="#FFFFE0">
<a href="javascript:NachLinks()">Nach links mit diesem Layer</a>
</layer>
</body></html>
```

Das Beispiel enthält einen Layer, beginnend bei Pixelposition 300 von links, mit einem Verweis. Beim Anklicken des Verweises wird die Funktion `NachLinks()` aufgerufen, die im Dateikopf in einem Script-Bereich notiert ist. Diese Funktion ändert den Links-Wert des Layers auf 0, sodass der Layer anschließend ganz links beginnt.

JS 1.2 N 4.0 Lesen Ändern *pageY*

Speichert den Pixelwert für oberhalb der linken oberen Ecke eines Layers. Bezug ist dabei das globale Dokument, auch wenn der aktuelle Layer innerhalb eines anderen Layers definiert ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  function NachOben()
  {
    document.Layerchen.pageY = 0;
  }
-->
</script>
</head><body>
<layer name="Layerchen" top="300" width="300" height="300" bgcolor="#FFFFE0">
<a href="javascript:NachOben()">Nach oben mit diesem Layer</a>
</layer>
</body></html>
```

```

    }
  //-->
</script>
</head><body>
<layer name="Layerchen" top="300" width="300" bgcolor="#FFFE0">
<a href="javascript:NachOben()">Nach oben mit diesem Layer</a>
</layer>
</body></html>

```

Das Beispiel enthält einen Layer, beginnend bei Pixelposition 300 von oben, mit einem Verweis. Beim Anklicken des Verweises wird die Funktion `NachOben()` aufgerufen, die im Dateikopf in einem Script-Bereich notiert ist. Diese Funktion ändert den Links-Wert des Layers auf 0, sodass der Layer anschließend ganz links beginnt.



### *parentLayer*

Speichert das Eltern-Objekt eines Layers. Das kann entweder das `window`-Objekt sein, oder, falls der Layer innerhalb eines anderen Layers definiert ist, dieser übergeordnete Layer. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<layer name="SuperLayer">
  <layer name="Layerchen" bgcolor="#FFFE0">
    <a
href="javascript:alert(document.SuperLayer.document.Layerchen.parentLayer.name)">
      Super-Name?</a>
    </layer>
  </layer>
</body></html>

```

Das Beispiel enthält einen Layer innerhalb eines anderen Layers. Beim Anklicken des Verweises im inneren Layer wird der Name des Eltern-Objekts ausgegeben. Da das Elternobjekt in diesem Fall der übergeordnete Layer ist, wird dessen Name, also `SuperLayer`, ausgegeben.



### *siblingAbove*

Speichert denjenigen Layer, der aufgrund der Schichtposition, die z.B. mit `z-index=` in HTML festgelegt wird, über dem angesprochenen Layer liegt. Es handelt sich dabei um ein Layer-Objekt, für das seinerseits alle verfügbaren Eigenschaften und Methoden gelten. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<layer name="L1" left="100" top="100" z-index="3" bgcolor="#FFFE0">Layerinhalt
L1</layer>
<layer name="L2" left="120" top="120" z-index="1" bgcolor="#FFFD0">Layerinhalt
L2</layer>
<layer name="L3" left="140" top="140" z-index="4" bgcolor="#FFFC0">Layerinhalt
L3</layer>
<layer name="L4" left="140" top="140" z-index="2" bgcolor="#FFFB0">Layerinhalt
L4</layer>
<layer name="L5" left="100" top="180">

```

```
<a href="javascript:alert(document.L1.siblingAbove.name)">L1 siblingAbove</a><br>
<a href="javascript:alert(document.L2.siblingAbove.name)">L2 siblingAbove</a><br>
<a href="javascript:alert(document.L3.siblingAbove.name)">L3 siblingAbove</a><br>
<a href="javascript:alert(document.L4.siblingAbove.name)">L4 siblingAbove</a><br>
</layer>
</body></html>
```

Das Beispiel definiert insgesamt fünf Layer. Der zuletzt definierte Layer enthält vier Verweise. Jeder Verweis gibt für einen der ersten vier Layer den Namen desjenigen Layers aus, der aufgrund der z-index-Ordnung über dem angesprochenen Layer liegt. Klickt man beispielsweise auf den ersten Verweis, wird für den Layer mit dem Namen L1 der Name des Layers ausgegeben, der darüber liegt. Das ist in dem Fall der Layer mit dem Namen L3, da der Layer L1 den z-index-Wert 3 hat und der Layer L3 den nächsthöheren z-index-Wert, nämlich 4.

### *siblingBelow*

Speichert denjenigen Layer, der aufgrund der Schichtposition, die z.B. mit z-index= in HTML festgelegt wird, unter dem angesprochenen Layer liegt. Es handelt sich dabei um ein Layer-Objekt, für das seinerseits alle verfügbaren Eigenschaften und Methoden gelten. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<layer name="L1" left="100" top="180">
<a href="javascript:alert(document.L2.siblingBelow.name)">L2 siblingBelow</a><br>
<a href="javascript:alert(document.L3.siblingBelow.name)">L3 siblingBelow</a><br>
<a href="javascript:alert(document.L4.siblingBelow.name)">L4 siblingBelow</a><br>
<a href="javascript:alert(document.L5.siblingBelow.name)">L5 siblingBelow</a><br>
</layer>
<layer name="L2" left="100" top="100" z-index="3" bgcolor="#FFFE0">Layerinhalt
L2</layer>
<layer name="L3" left="120" top="120" z-index="1" bgcolor="#FFFD0">Layerinhalt
L3</layer>
<layer name="L4" left="140" top="140" z-index="4" bgcolor="#FFFC0">Layerinhalt
L4</layer>
<layer name="L5" left="140" top="140" z-index="2" bgcolor="#FFFB0">Layerinhalt
L5</layer>
</body></html>
```

Das Beispiel definiert insgesamt fünf Layer. Der erste Layer enthält vier Verweise. Jeder Verweis gibt für einen der folgenden vier Layer den Namen desjenigen Layers aus, der aufgrund der z-index-Ordnung unter dem angesprochenen Layer liegt. Klickt man beispielsweise auf den ersten Verweis, wird für den Layer mit dem Namen L2 der Name des Layers ausgegeben, der darunter liegt. Das ist in dem Fall der Layer mit dem Namen L5, da der Layer L1 den z-index-Wert 3 hat und der Layer L2 den nächstniedrigeren z-index-Wert, nämlich 2.

### Lesen *src*

Speichert die externe Datei, die in einen Layer eingebunden ist. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<layer name="extern" left="50" top="30" width="600" height="200" src="news.htm">
```

```

</layer>
<layer left="50" top="250">
<script type="text/javascript">
<!--
document.write("<hr><b>Eingebunden wurde die Datei: </b>" +
window.document.extern.src);
//-->
</script>
</layer>
</body></html>

```

Das Beispiel enthält zwei positionierte Layer. Der Inhalt des ersten Layers ist eine externe Datei namens *news.htm*. Im zweiten Layer wird mit Hilfe von `document.write()` die genaue Adresse der externen Datei in den Layerbereich geschrieben. Beachten Sie, dass in diesem Fall `window.document.extern.src` zur Adressierung des anderen Layers verwendet werden muss, da `document.extern.src` sich sonst nur auf den aktuellen Layer beziehen würde und dieser kein Element namens `extern` enthält.



Speichert den Pixelwert für oberhalb der linken oberen Ecke eines Layers. Bezug ist dabei das übergeordnete Dokument. Das kann entweder das globale Dokument sein oder ein Layer, innerhalb dessen der aktuelle Layer definiert ist. Für absoluten Bezug siehe `pageY`. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<layer name="SuperLayer" top="100">
  <layer name="Layerchen" top="200" width="300" bgcolor="#FFFFE0">
    <a href="javascript:alert(document.SuperLayer.document.Layerchen.top)">Oben-Wert?</a>
  </layer>
</layer>
</body></html>

```

Das Beispiel enthält einen Layer innerhalb eines anderen Layers, beginnend bei Pixelposition 200 von oben, mit einem Verweis. Beim Anklicken des Verweises wird der Oben-Wert des inneren Layers ausgegeben. Der beträgt 200, obwohl der Layer optisch gesehen bei Position 300 beginnt, weil der übergeordnete Layer ja schon bei 100 beginnt und die 200 des inneren Layers dazu addiert werden.



Speichert die Information, ob bzw. dass ein Layer sichtbar ist oder nicht. Im Beispiel werden zwei positionierte Layer definiert. Jeder der beiden Layer enthält einen Verweis, bei dessen Anklicken der jeweils andere Layer aus- und wieder eingeblendet wird. Dazu wird jeweils die Funktion `EinAus()` aufgerufen, die im Dateikopf notiert ist. Als Parameter wird die Indexnummer des jeweils anderen Layers übergeben. Die Eigenschaft `visibility` kann die drei Werte `hide` (*verstecken*), `show` (*anzeigen*) und `inherit` (*wie vom Eltern-Layer vererbt*) haben.

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--

```



```
function EinAus(Nummer)
{
    if(window.document.layers[Nummer].visibility == "show")
    { window.document.layers[Nummer].visibility = "hide"; return; }
    if(window.document.layers[Nummer].visibility == "hide")
    { window.document.layers[Nummer].visibility = "show"; return; }
}
//-->
</script>
</head><body>
<layer name="GelberLayer" left="50" top="30" width="200" height="200"
    visibility="show" bgcolor="#FFFFE0">
<a href="javascript:EinAus(1)">anderer Layer ein/aus</a>
</layer>
<layer name="LilaLayer" left="250" top="30" width="200" height="200"
    visibility="show" bgcolor="#FFE0FF">
<a href="javascript:EinAus(0)">anderer Layer ein/aus</a>
</layer>
</body></html>
```

### JS 1.2 N 4.0 Lesen Ändern **zIndex**

Speichert die Schichtposition eines Layers. Im Beispiel wird ein Layer definiert, der per HTML-Attribut einen z-index-Wert von 527 erhält. Innerhalb des Layers ist ein Verweis notiert. Beim Anklicken des Verweises wird der z-Index-Wert des Layers in einem Meldungsfenster ausgegeben.

```
<html><head><title>Test</title>
</head><body>
<layer z-index="527" name="EinLayer">
<a href="javascript:alert(window.document.EinLayer.zIndex)">z-index?</a>
</layer>
</body></html>
```

### JS 1.2 N 4.0 **captureEvents()**

Überwacht Anwenderereignisse in einem Layer. Funktioniert genau so wie `captureEvents()` beim `window`-Objekt (nähere Informationen siehe Kapitel 20.3). Der einzige Unterschied ist, dass Sie mit `window.document.Layername.captureEvents()` nur Ereignisse innerhalb des Layerbereichs überwachen können, keine Ereignisse im gesamten Fensterbereich (zu dem beispielsweise auch Titelleiste, Menüleisten usw. gehören).

### JS 1.2 N 4.0 **handleEvent()**

Übergibt ein Ereignis an ein Element, das in der Lage ist, auf das Ereignis zu reagieren. Funktioniert genauso wie `handleEvent()` beim `window`-Objekt (nähere Informationen siehe Kapitel 20.3). Der einzige Unterschied ist, dass `window.document.Layername.handleEvent()` nur Ereignisse übergeben kann, die mit `window.document.Layername.captureEvents()` überwacht werden.

**JS**  
**1.2** **N**  
**4.0** **load()**

Lädt eine externe Datei in einen Layer. Erwartet zwei Parameter:

- *Quelle* = (HTML)-Datei, die geladen werden soll.
- *Breite* = Anzeigebreite des Layers.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
    function Laden()
    {
        document.EinLayer.load("news.htm",600);
    }
//-->
</script>
</head><body>
<a href="javascript:Laden()">News laden</a>
<layer name="EinLayer" top="50" left="20">
</layer>
</body></html>
```

Das Beispiel enthält einen leeren Layer und einen Verweis. Beim Anklicken des Verweises wird die Funktion `Laden()` aufgerufen, die im Dateikopf notiert ist. Die Anweisung innerhalb der Funktion wendet die Methode `load()` auf den bis dahin leeren Layer an. Die angegebene Datei *news.htm* wird in den Layer geladen, und der Layer erhält eine Anzeigebreite von 600 Pixeln.

**JS**  
**1.2** **N**  
**4.0** **moveAbove()**

Legt einen Layer über einen anderen Layer. Erwartet als Parameter den Layer, der überdeckt werden soll. Die Schichtposition des veränderten Layers ändert sich dabei abhängig von der des Layers, der überdeckt werden soll. Diese Methode hat nichts mit dem Bewegen von Layern zu tun. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<layer name="GelberLayer" top="150" left="100" width="200" height="200"
bgColor="#FFFFE0">
Gelber Layer<br>
<a href="javascript:window.document.GelberLayer.moveAbove(window.document.LilaLayer)">
nach vorne</a>
</layer>
<layer name="LilaLayer" top="50" left="250" width="200" height="200"
bgColor="#FFE0FF">
Lila Layer<br>
<a href="javascript:window.document.LilaLayer.moveAbove(window.document.GelberLayer)">
nach vorne</a>
</layer>
</body></html>
```

Im Beispiel werden zwei positionierte Layer definiert, die sich zum Teil überlappen. Beide enthalten jeweils einen Verweis, um den eigenen Layer über den anderen zu legen.

### **moveBelow()**

Legt einen Layer unter einen anderen Layer. Erwartet als Parameter den Layer, der den aktuellen überdecken soll. Die Schichtposition des veränderten Layers ändert sich dabei abhängig von der des Layers, der den aktuellen Layer überdecken soll. Diese Methode hat nichts mit dem Bewegen von Layern zu tun. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<layer name="GelberLayer" top="150" left="100" width="200" height="200"
bgColor="#FFFFE0">
Gelber Layer<br>
<a href="javascript:window.document.GelberLayer.moveBelow(window.document.LilaLayer)">
nach hinten</a>
</layer>
<layer name="LilaLayer" top="50" left="250" width="200" height="200"
bgColor="#FFE0FF">
Lila Layer<br>
<a href="javascript:window.document.LilaLayer.moveBelow(window.document.GelberLayer)">
nach hinten</a>
</layer>
</body></html>
```

Im Beispiel werden zwei positionierte Layer definiert, die sich zum Teil überlappen. Beide enthalten jeweils einen Verweis, um den eigenen Layer von dem jeweils anderen überdecken zu lassen.

### **moveBy()**

Verschiebt einen Layer um so viele Pixel wie angegeben. Erwartet folgende Parameter:

- *x-Wert* = Anzahl Pixel, um die der Layer nach links/rechts verschoben werden soll. Negative Werte verschieben nach links, positive nach rechts.
- *y-Wert* = Anzahl Pixel, um die der Layer nach oben/unten verschoben werden soll. Negative Werte verschieben nach oben, positive nach unten.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Springen()
{
var x = parseInt(document.Springer.document.Eingabe.hr.value);
var y = parseInt(document.Springer.document.Eingabe.vr.value);
document.Springer.moveBy(x,y);
}
//-->
```

```

</script>
</head><body>
<layer name="Springer" top="150" left="100" width="200" height="200"
bgColor="#FFFFE0">
<form name="Eingabe" action="">
Springe um <input size="5" name="hr"> Pixel horizontal<br>
Springe um <input size="5" name="vr"> Pixel vertikal<br>
<input type="button" value="Springe" onClick="Springen()">
</form>
</body></html>

```

Im Beispiel wird ein Layer definiert, der ein Formular enthält. Das Formular stellt zwei kleine Eingabefelder bereit, in denen der Anwender eingeben kann, um wie viele Pixel der Layer horizontal und vertikal verschoben werden soll. Beim Anklicken des Buttons, der auch zu dem Formular gehört, wird die Funktion `Springen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ermittelt zunächst die eingegebenen Werte. Da es sich um Zeichenketten handelt, die Methode `moveBy()` aber Zahlen erwartet, wird die Funktion `parseInt()` zum Umwandeln der Formulareingaben in eine Zahl angewendet. Anschließend wendet die Funktion die Methode `moveBy()` mit den ermittelten Werten auf den Layer an.

### **moveTo()**

Verschiebt einen Layer auf die anzugebende Position. Erwartet folgende Parameter:

- *x-Wert* = Links-Wert für die neue obere linke Ecke des Fensters.
- *y-Wert* = Oben-Wert für die neue obere linke Ecke des Fensters.

Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Springen()
{
var x = parseInt(document.Springer.document.Eingabe.hr.value);
var y = parseInt(document.Springer.document.Eingabe.vr.value);
document.Springer.moveTo(x,y);
}
//-->
</script>
</head><body>
<layer name="Springer" top="150" left="100" width="200" height="200"
bgColor="#FFFFE0">
<form name="Eingabe" action="">
links oben neu: <input size="5" name="hr"> (von links)<br>
links oben neu: <input size="5" name="vr"> (von oben)<br>
<input type="button" value="Springe" onClick="Springen()">
</form>
</body></html>

```

Im Beispiel wird ein Layer definiert, der ein Formular enthält. Das Formular stellt zwei kleine Eingabefelder bereit, in denen der Anwender die x- und y-Werte der neuen gewünschten oberen linken Ecke des Layers eingeben kann. Beim Anklicken des Buttons, der auch zu dem Formular gehört, wird die Funktion `Springen()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ermittelt zunächst die eingegebenen Werte. Da es sich um Zeichenketten handelt, die Methode `moveTo()` aber Zahlen erwartet, wird die Funktion `parseInt()` zum Umwandeln der Formulareingaben in eine Zahl angewendet. Anschließend wendet die Funktion die Methode `moveTo()` mit den ermittelten Werten auf den Layer an.

**Beachten Sie:** Wenn sich der Layer, auf den Sie die `moveTo()`-Methode anwenden, innerhalb eines anderen Layers befindet, beziehen sich die Angaben relativ zu dem äußeren Layer. Eine Angabe wie `moveTo(0,0)` springt also zur linken oberen Ecke des äußeren Layers.

### ***moveToAbsolute()***

Funktioniert genauso wie `moveTo()`, mit dem Unterschied, dass die x- und y-Werte absolut zum Anzeigefenster interpretiert werden, falls sich der Layer innerhalb eines anderen Layers befindet.

### ***releaseEvents()***

Beendet die Überwachung von Anwenderereignissen. Funktioniert genau so wie `releaseEvents()` beim `window`-Objekt (nähere Informationen siehe dort). Der einzige Unterschied ist, dass `document.Layername.releaseEvents()` nur Ereignisüberwachungen beenden kann, die mit `document.Layername.captureEvents()` gestartet wurden.

### ***resizeBy()***

Verändert die Größe eines Layers um so viele Pixel unten und rechts wie angegeben. Erwartet folgende Parameter:

- *x-Wert* = Anzahl Pixel, um die der Layer rechts verkleinert/vergrößert werden soll. Negative Werte verkleinern, positive vergrößern.
- *y-Wert* = Anzahl Pixel, um die der Layer unten verkleinert/vergrößert werden soll. Negative Werte verkleinern, positive vergrößern.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function MachKleiner() {
  document.MeinLayer.resizeBy(-20,-10);
}
function MachGroesser() {
  document.MeinLayer.resizeBy(20,10);
}
//-->
</script>
</head><body>
```

```
<layer name="MeinLayer" top="50" left="50" width="200" height="100"
bgColor="#FFFFE0">
<a href="javascript:MachKleiner()">Kleiner</a><br>
<a href="javascript:MachGroesser()">Groesser</a><br>
</layer>
</body></html>
```

Im Beispiel ist ein Layer definiert. Mit Hilfe von zwei Verweisen lässt sich im Beispiel der sichtbare Anzeigebereich des Layers verkleinern und wieder vergrößern. Die Verweise rufen die Funktionen `MachKleiner()` bzw. `MachGroesser()` auf, die in einem Script-Bereich im Dateikopf notiert sind. Die Funktion `MachKleiner()` verringert die Breite des Layers um 20 Pixel und die Höhe um 10 Pixel, die Funktion `MachGroesser()` tut das Umgekehrte.



### **resizeTo()**

Legt die Breite und Höhe eines Layers neu fest. Erwartet folgende Parameter:

- *x-Wert* = neue Breite des Layers.
- *y-Wert* = neue Höhe des Layers.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Aendern()
{
var x = parseInt(document.L.document.Eingabe.hr.value);
var y = parseInt(document.L.document.Eingabe.vr.value);
document.L.resizeTo(x,y);
}
//-->
</script>
</head><body>
<layer name="L" top="50" left="50" width="200" height="200" bgColor="#FFFFE0">
<form name="Eingabe" action="">
neue <input size="5" name="hr"> Breite<br>
neue <input size="5" name="vr"> Hoehe<br>
<input type="button" value="Aendern" onClick="Aendern()">
</form>
</body></html>
```

Im Beispiel wird ein Layer definiert, der ein Formular enthält. Das Formular stellt zwei kleine Eingabefelder bereit, in denen der Anwender die x- und y-Werte der neuen gewünschten Breite und Höhe des Layers eingeben kann. Beim Anklicken des Buttons, der auch zu dem Formular gehört, wird die Funktion `Aendern()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion ermittelt zunächst die eingegebenen Werte. Da es sich um Zeichenketten handelt, die Methode `resizeTo()` aber Zahlen erwartet, wird die Funktion `parseInt()` zum Umwandeln der Formulareingaben in eine Zahl angewendet. Anschließend wendet die Funktion die Methode `resizeTo()` mit den ermittelten Werten auf den Layer an.



**routeEvent()**

Übergibt ein Anwenderereignis an das `document`-Objekt. Von dort aus wird das Ereignis immer weiter nach unten durchgereicht, bis sich ein Event-Handler findet, der das Ereignis verarbeitet. Funktioniert genauso wie `routeEvent()` beim `window`-Objekt (nähere Informationen siehe dort). Der einzige Unterschied ist, dass `document.Layername.routeEvent()` nicht auf Fenster-ebene ansetzt, sondern auf Layerebene.

## 20.17 links

### Eigenschaft

|                     |                                      |
|---------------------|--------------------------------------|
| <code>name</code>   | (Name des Verweises)                 |
| <code>length</code> | (Anzahl Verweise)                    |
| <code>target</code> | (Zielfenster des Verweises)          |
| <code>text</code>   | (Text des Verweises)                 |
| <code>x</code>      | (horizontale Position des Verweises) |
| <code>y</code>      | (vertikale Position des Verweises)   |

### links: Allgemeines zur Verwendung

Mit dem Objekt `links`, das in der JavaScript-Objekthierarchie unterhalb des `document`-Objekts liegt, haben Sie Zugriff auf Verweise, die in einer HTML-Datei definiert sind. In JavaScript 1.0 ist es lediglich möglich, die Anzahl der Verweise in einer Datei auszulesen. Seit JavaScript 1.2 stehen jedoch weitere Eigenschaften zur Verfügung.




**Zugriff auf Verweisziele**

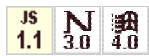
Sie können mit JavaScript die Verweisziele von Verweisen in HTML ermitteln. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="http://www.yahoo.de/">Yahoo</a><br>
<a href="http://www.google.de/">Google</a><br>
<a href="http://www.teamone.de/">TeamOne</a><br>
<script type="text/javascript">
<!--
for(var i=0; i < document.links.length; ++i)
  document.write("<br>" + document.links[i]);
//-->
</script>
</body></html>
```

Im Beispiel werden ein paar Verweise notiert. Unterhalb davon steht ein JavaScript, das mit `document.write()` in einer `for`-Schleife für jeden Verweis das Verweisziel in die Datei schreibt. Es handelt sich um denjenigen Wert, der bei den Verweisen hinter `href=` notiert ist.

Der Zugriff auf die Verweise erfolgt mit Indexnummern. Dabei geben Sie `document.links` an und dahinter in eckigen Klammern, den wievielten Verweis in der Datei Sie meinen. Beachten

Sie, dass der Zähler bei 0 beginnt, d.h., den ersten Verweis sprechen Sie mit `links[0]` an, den zweiten Verweis mit `links[1]` usw. Beim Zählen gilt die Reihenfolge, in der die Verweise in der Datei notiert sind. Dabei werden auch Verweise innerhalb von verweissensitiven Grafiken (`<area>`-Tag) berücksichtigt.



### location-Eigenschaften von Verweisen

Neben den verweisspezifischen Eigenschaften kennt das `Links`-Objekt auch alle Eigenschaften des `location`-Objekts. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a target="_top" href="http://www.teamone.de/">TeamOne</a><br>
<script type="text/javascript">
<!--
document.links[0].port="8081";
//-->
</script>
</body></html>
```

Über das Objekt `location` haben Sie Zugriff auf den vollständigen URI eines angezeigten Verweises. Sie können den URI oder Teile davon zur Weiterverarbeitung abfragen und ändern. Notieren Sie dazu das `Links`-Objekt, also z.B. `document.links[0]`, dahinter einen Punkt und anschließend die gewünschte Eigenschaft des `location`-Objekts. Im Beispiel ist ein Verweis definiert. Mit `document.links[0].port=` wird die Eigenschaft `port` geändert.

**Beachten Sie:** Im Netscape 6.1 und in Opera 5.12 können Sie die Eigenschaften des `location`-Objekts (mit Ausnahme der `href`-Eigenschaft) nur lesen, jedoch nicht ändern.



### name

Speichert den Namen eines Verweises. Durch Verwendung des Attributes `name=` wird der Verweis gleichzeitig zu einem Bestandteil des `anchors`-Objektes. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a name="katalog" href="http://www.yahoo.de/">Yahoo</a><br>
<a name="suchmaschine" href="http://www.google.de/">Google</a><br>
<a name="redaktion" href="http://www.teamone.de/">TeamOne</a><br>
<script type="text/javascript">
<!--
for(var i=0; i < document.links.length; ++i)
document.write("<br>" + document.links[i].name);
//-->
</script>
</body></html>
```

Mit `document.links[0].name` können Sie den Namen eines Verweises in einer HTML-Datei ermitteln. Das Beispiel enthält zunächst HTML-Text mit diversen Verweisen und schreibt am Ende mit `document.write()` die Namen der Verweise ins Dokument.



**Beachten Sie:** Gegenwärtig ist es nur im Netscape 6.1 möglich, den Namen eines Verweises zu ändern.



Speichert, wie viele Verweise eine HTML-Datei enthält. Mit `document.links.length` können Sie die Anzahl der Verweise in einer HTML-Datei ermitteln. Das Beispiel enthält zunächst HTML-Text mit diversen Verweisen und schreibt am Ende mit `document.write()` dazu, wie viele Verweise in der Datei gefunden wurden. Beachten Sie, dass dabei (seit Netscape 3.0) auch Verweise berücksichtigt werden, die in verweissensitiven Grafiken mit Hilfe des `<area>`-Tags definiert werden.

```
<html><head><title>Test</title>
</head><body>
<a href="http://www.yahoo.de/">Yahoo</a><br>
<a href="http://www.lycos.de/">Lycos</a><br>
<map name="Testbild">
<area shape="rect" coords="1,1,249,49" href="#Ziel_1" alt="">
<area shape="rect" coords="1,51,149,299" href="#Ziel_2" alt="">
</map>

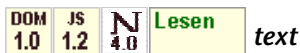
<script type="text/javascript">
<!--
document.write("<p>Die Datei hat " + document.links.length + " Verweise</p>");
//-->
</script>
</body></html>
```



Speichert das Zielfenster eines Verweises. Mit `document.links[0].target` können Sie das Zielfenster eines Verweises ermitteln. Das Beispiel enthält einen Verweis, der sein Verweisziel im obersten Fenster öffnet. Im nachfolgenden JavaScript-Bereich wird diesem Verweis mit `document.links[0].target="_blank"` ein anderes Zielfenster zugewiesen. Das führt dazu, dass sich das Ziel des Verweises in einem neuen Fenster öffnet.

```
<html><head><title>Test</title>
</head><body>
<a href="http://www.teamone.de/" target="_top">TeamOne</a><br>
<script type="text/javascript">
<!--
document.links[0].target="_blank";
//-->
</script>
</body></html>
```

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel nicht.



Speichert den Text eines Verweises. Mit `document.anchors[0].text` können Sie den Text eines Verweises zwischen `<a>` und `</a>` ermitteln. Das Beispiel enthält zunächst einige Verweise. Unterhalb davon steht ein JavaScript, das mit `document.write()` in einer `for`-Schleife für jeden Verweis den Verweistext in die Datei schreibt.

```
<html><head><title>Test</title>
</head><body>
<a href="http://www.yahoo.de/">Yahoo</a><br>
<a href="http://www.google.de/">Google</a><br>
<a href="http://www.teamone.de/">TeamOne</a><br>
<script type="text/javascript">
<!--
for(var i=0; i < document.links.length; ++i)
  document.write("<br>" + document.links[i].text);
//-->
</script>
</body></html>
```



Speichert die horizontale Position eines Verweises gemessen vom Fensterrand. Mit `document.links[0].x` können Sie die horizontale Position eines Verweises in einer HTML-Datei ermitteln. Das Beispiel enthält einen kleinen Text, in dem ein Verweis definiert ist. Beim Anklicken des Verweises wird mit `alert()` die Position des Verweises vom linken Fensterrand aus gesehen ausgegeben.

```
<html><head><title>Test</title>
</head><body>
Hier ein Text und dann ein <a
href="javascript:alert(document.links[0].x)">Verweis</a><br>
</body></html>
```



Speichert die vertikale Position eines Verweises gemessen vom Fensterrand. Mit `document.links[0].y` können Sie die horizontale Position eines Verweises in einer HTML-Datei ermitteln. Das Beispiel enthält einen kleinen Text, in dem ein Verweis definiert ist. Beim Anklicken des Verweises wird mit `alert()` die Position des Verweises vom oberen Fensterrand aus gesehen ausgegeben.

```
<html><head><title>Test</title>
</head><body>
Hier ein Text und dann ein <a
href="javascript:alert(document.links[0].y)">Verweis</a><br>
</body></html>
```

## 20.18 event

### Eigenschaften

|                                        |                                                |
|----------------------------------------|------------------------------------------------|
| <code>altKey, ctrlKey, shiftKey</code> | (Sondertasten/Microsoft)                       |
| <code>clientX, clientY</code>          | (Bildschirmkoordinaten/Microsoft)              |
| <code>keyCode</code>                   | (Tastaturcode/Microsoft)                       |
| <code>layerX, layerY</code>            | (Objekt-relative Koordinaten/Netscape)         |
| <code>modifiers</code>                 | (Sondertasten/Netscape)                        |
| <code>offsetX, offsetY</code>          | (Objekt-relative Koordinaten/Microsoft)        |
| <code>pageX, pageY</code>              | (Fenster-relative Koordinaten/Netscape)        |
| <code>screenX, screenY</code>          | (Bildschirmkoordinaten/Netscape)               |
| <code>which</code>                     | (Tastatur-/Maustastencode/Netscape)            |
| <code>type</code>                      | (Art des Ereignisses/Netscape)                 |
| <code>x, y</code>                      | (Elternelement-relative Koordinaten/Microsoft) |

### event: Allgemeines zur Verwendung

Mit dem Objekt `event` können Sie Einzelinformationen zu Anwenderereignissen wie Mausklicks oder Tasteneingaben ermitteln und weiterverarbeiten. So können Sie bei einem Mausklick beispielsweise die genaue Position ermitteln, wo der Mausklick erfolgte, oder bei einem Tastendruck die gedrückte Taste abfragen.

Anwenderereignisse können Sie entweder überwachen, indem Sie in einem erlaubten HTML-Tag einen Event-Handler notieren, oder, indem Sie direkt mit Hilfe von JavaScript eine Ereignisüberwachung programmieren. Für den Fall, dass das überwachte Ereignis eintritt, können Sie eine Handler-Funktion schreiben, die das Ereignis »behandelt«, also verarbeitet. Die Handler-Funktion wird automatisch aufgerufen, wenn das Ereignis eintritt. Innerhalb einer Handler-Funktion besteht auch die Möglichkeit, Eigenschaften des eingetretenen Ereignisses abzufragen. Solche Eigenschaften werden auf dieser Seite hier beschrieben.

Sowohl Netscape 4.x als auch der MS Internet Explorer 4.x kennen das `event`-Objekt. Leider ist die Implementierung jedoch völlig unterschiedlich gelöst und führt zu einer ziemlich chaotischen Situation in diesem Bereich. Die Eigenschaften des `event`-Objekts, die auf dieser Seite beschrieben werden, berücksichtigen beide Browser. In der Regel ist eine Eigenschaft jedoch entweder nur bei Netscape oder nur beim MS Internet Explorer verfügbar. Auch in der Syntax zur Überwachung von Ereignissen unterscheiden sich beide Browser. Das gilt sowohl bei der Ereignisüberwachung per Event-Handler in HTML als auch für die direkte Ereignisüberwachung in JavaScript. Die Unterschiede bei den Event-Handlern werden im Abschnitt über Event-Handler näher beschrieben. Hier wird zur allgemeinen Verwendung des `event`-Objekts beschrieben, wie Sie mit Hilfe von JavaScript eine direkte Ereignisüberwachung programmieren. Das Beispiel zeigt auch, wie Sie eine solche Ereignisüberwachung für beide Browser getrennt programmieren können, ohne dass es zu Fehlermeldungen kommt.

Beispiel (Ereignisüberwachung direkt mit JavaScript programmieren):

```
<html><head><title>Test</title>

<script language="JavaScript1.2" type="text/javascript">
<!--
```

```

var Netscape = new Boolean();
if(navigator.appName == "Netscape") Netscape = true;

function TasteGedrueckt(Ereignis)
{
    if(Netscape)
    { window.status = "Taste mit Dezimalwert " + Ereignis.which + " gedrueckt";
    return true; }
}
function TasteLosgelassen(Ereignis)
{
    if(Netscape)
    { window.status = "Taste mit Dezimalwert " + Ereignis.which + " losgelassen";
    return true; }
}
document.onkeydown = TasteGedrueckt;
document.onkeyup = TasteLosgelassen;

```

Es werden zwei Funktionen definiert: Die Funktion `TasteGedrueckt(Ereignis)` ist die so genannte Handler-Funktion für den Fall, dass der Anwender eine Taste drückt. Die Funktion `TasteLosgelassen(Ereignis)` behandelt den Fall, dass der Anwender die gedrückte Taste wieder loslässt. Beide Funktionen erwarten einen Parameter namens `Ereignis`. Über diesen Parameter können die Funktionen auf Eigenschaften des `event`-Objekts zugreifen. In den beiden Funktionen im Beispiel wird die Objekteigenschaft `which` ausgewertet, die nach Netscape-Syntax den dezimalen Tastaturcode einer gedrückten Taste speichert.

Das allein genügt jedoch noch nicht zur Ereignisüberwachung. Damit die Handler-Funktionen bei Eintritt des Ereignisses automatisch aufgerufen werden, muss die Ereignisüberwachung gestartet werden. Dazu dienen die beiden Anweisungen am Ende des JavaScript-Bereichs.

Mit `document.onkeydown = TasteGedrueckt;` wird das Ereignis »Taste gedrückt« (`onkeydown`) überwacht. Hinter dem Gleichheitszeichen folgt der Name der Handler-Funktion, die dieses Ereignis verarbeiten soll (`TasteGedrueckt`). Beachten Sie, dass hier nur der Funktionsname stehen darf, kein Funktionsaufruf mit runden Klammern dahinter.

Mit `document.onkeyup = TasteLosgelassen;` wird das Ereignis »Taste losgelassen« (`onkeyup`) überwacht, und zur Ereignisbehandlung wird die Handler-Funktion `TasteLosgelassen` angegeben. Nach dem gleichen Schema können Sie auch andere Ereignisse überwachen – die Handler-Funktionen definieren Sie selber, und anstelle von `onkeydown` und `onkeyup` können Sie alle anderen üblichen Event-Handler notieren.

Bei den beiden Script-Bereichen für den MS Internet Explorer wird für jedes zu überwachende Ereignis ein Script-Bereich notiert. Mit einer Angabe wie `for="document" event="onkeydown()"` im einleitenden `<script>`-Tag geben Sie an, welches Ereignis Sie überwachen wollen (z.B. `onkeydown`), und für welches Objekt (z.B. `document`). Es sind alle üblichen Event-Handler erlaubt.

Innerhalb des Scripts können Sie dann auf Eigenschaften des `event`-Objekts zugreifen, die der MS Internet Explorer kennt. Mit `window.event.keyCode` wird beispielsweise der dezimale Tastaturcode einer gedrückten Taste ermittelt.

**Beachten Sie:** Das vorliegende Beispiel wird auch vom Netscape 6 entsprechend der Netscape-spezifischen Syntax interpretiert.



### *altKey, ctrlKey, shiftKey*

Microsoft-Syntax. Speichert, ob Zusatztasten wie die `[Alt]`-Taste oder die `[⇧]`-Taste gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt wurden. Ein Beispiel:

```
<html><head><title>Test</title>
<script for="document" event="onkeypress()" language="JScript" type="text/jscript">
<!--
{
  if(window.event.shiftKey)
    alert("eine Taste plus Umschalttaste gedrueckt!");
}
//-->
</script>
</head><body>
</body></html>
```

Im Beispiel wird überwacht, ob der Anwender eine Taste drückt (`onkeypress`). Wenn ja, wird abgefragt, ob zusätzlich die Umschalttaste gedrückt wurde. In diesem Fall wird eine entsprechende Meldung ausgegeben.



Lesen

***clientX, clientY***

Microsoft-Syntax. Speichert die horizontalen Pixel (`clientX`) und die vertikalen Pixel (`clientY`) der Cursor-Position relativ zur oberen linken Ecke des Anzeigefensters, wenn z.B. Mausereignisse überwacht werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script for="document" event="onmousedown()" language="JScript"
type="text/jscript">
<!--
{
  document.all.Springer.style.left = window.event.clientX;
  document.all.Springer.style.top = window.event.clientY;
}
//-->
</script>
</head><body>
<div id="Springer" style="background-color:#FFE0FF; position:absolute;
top:100px; left:100px; width:100px; height:100px;"></div>
</body></html>
```

Das Beispiel enthält einen `<div>`-Bereich, der mit Hilfe von Stylesheet-Angaben absolut positioniert wird. Wenn der Anwender mit der Maus in das Fenster klickt, wird der Bereich an die Position verschoben, an der der Mausklick erfolgt.



Lesen

***keyCode***

Microsoft-Syntax. Speichert bei Tastaturereignissen den dezimalen Code (ASCII/ANSI-Wert) der gedrückten Taste. Das Beispiel überwacht, ob bei aktivem Dokument eine Taste gedrückt wurde. Wenn ja, wird deren Wert in einem Meldungsfenster ausgegeben.

```
<html><head><title>Test</title>
<script for="document" event="onmousedown()" language="JScript"
type="text/jscript">
<!--
{
  alert(window.event.keyCode);
}
//-->
</script>
</head><body>
</body></html>
```

**JS** **N** **Lesen** *layerX, layerY*

Netscape-Syntax. Speichert die aktuelle Breite (*layerX*) und Höhe (*layerY*) eines Objekts, wenn das Ereignis *onresize* überwacht wird, oder die horizontalen Pixel (*layerX*) und die vertikalen Pixel (*layerY*) der Cursor-Position relativ zur oberen linken Ecke eines Layer-Bereichs, wenn z.B. Mausereignisse überwacht werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function wResize(Ereignis)
{
  window.status = Ereignis.layerX + "x" + Ereignis.layerY; return true;
}
window.onresize = wResize;
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel überwacht, ob der Anwender das Anzeigefenster des Browsers verändert. Beim Verändern der Größe des Anzeigefensters wird so oft wie möglich die Handler-Funktion *wResize* aufgerufen. Diese schreibt in die Statuszeile des Browsers die aktuelle Fenstergröße.

**Beachten Sie:** Netscape 6.1 interpretiert zwar den Event-Handler, jedoch ist in den Eigenschaften *layerX* und *layerY* jeweils der Wert 0 gespeichert.

**JS** **N** **Lesen** *modifiers*

Netscape-Syntax. Speichert, ob Zusatz Tasten wie die **Alt**-Taste oder die **Ctrl**-Taste gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt wurden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function ALT_Wert(Ereignis)
{
  if(Ereignis.modifiers & Event.ALT_MASK)
    alert("Maus geklickt und ALT-Taste gedrueckt!")
}
document.onmouseup = ALT_Wert;
//-->
</script>
</head><body>
</body></html>
```

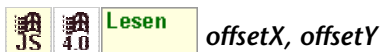
Im Beispiel wird das Meldungsfenster dann angezeigt, wenn der Anwender mit der Maus in das Dokument geklickt hat und gleichzeitig die **Alt**-Taste gedrückt hält. Die verwendete Syntax benutzt Bit-Operatoren. Fragen Sie einfach so ab wie in dem Beispiel. Folgende Ausdrücke sind erlaubt:

- `Ereignis.modifiers & Event.ALT_MASK`, um die **Alt**-Taste abzufragen;

- Ereignis.modifiers & Event.CONTROL\_MASK, um die **Ctrl**-Taste (**Strg**-Taste) abzufragen;
- Ereignis.modifiers & Event.SHIFT\_MASK, um die **⇧**-Taste (Großschreibung) abzufragen;
- Ereignis.modifiers & Event.META\_MASK, um die **AltGr**-Taste abzufragen.

Diese Angaben sind natürlich abhängig davon, ob die verwendete Tastatur eine solche Taste überhaupt enthält.

**Beachten Sie:** Netscape 6.1 interpretiert dieses Beispiel nicht.

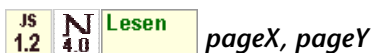


### *offsetX, offsetY*

Microsoft-Syntax. Speichert die horizontalen Pixel (`clientX`) und die vertikalen Pixel (`clientY`) der Cursor-Position relativ zur oberen linken Ecke des Elements, das ein Ereignis ausgelöst hat. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
function Coords()
{
    alert("Stelle im Button: " + window.event.offsetX + "/" + window.event.offsetY);
}
//-->
</script>
</head><body>
<form action="">
<input type="button" value="Klick doch mal auf mich drauf" onClick="Coords()">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einem Button. Beim Anklicken des Buttons wird die Funktion `Coords()` aufgerufen. Diese gibt die Pixelkoordinaten des Mausklicks relativ zur oberen linken Ecke des Elements, also des Buttons, aus.



### *pageX, pageY*

Netscape-Syntax. Speichert die horizontalen Pixel (`pageX`) und die vertikalen Pixel (`pageY`) der Cursor-Position relativ zur oberen linken Ecke der Seite, wenn z.B. Mausereignisse überwacht werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function LayerPos(Ereignis)
{
    if (document.layers) {
        document.layers[0].left = Ereignis.pageX;
        document.layers[0].top = Ereignis.pageY;
    }
}
```



```

else if (window.netscape) {
    document.getElementsByTagName("div")[0].style.left = Ereignis.pageX;
    document.getElementsByTagName("div")[0].style.top = Ereignis.pageY;
}
}
document.onmouseup = LayerPos;
//-->
</script>
</head><body>
<div style="position:absolute;top:50px;left:50px;width:100px;height:100px;
    background-color:#FFE0FF;border:solid 1px #000000;">
Ein Layer<br><br>
</div>
</body></html>

```

Das Beispiel enthält einen simulierten Layer. Wenn der Anwender mit der Maus in das Fenster klickt und die Maustaste dann wieder loslässt, wird der Layer an die Position verschoben, an der die Maus losgelassen wurde.

Da die Eigenschaften `pageX` und `pageY` auch von Netscape 6 interpretiert werden, wurde das Beispiel entsprechend kompatibel gestaltet. Statt eines Layers (Netscape 6 kennt keine Layer mehr) wird mit einem absolut positionierten `div`-Bereich ein Layer ohne `layer`-Element simuliert. Wenn die Maustaste gedrückt und wieder losgelassen wird, wird die Funktion `LayerPos()` aufgerufen. Innerhalb dieser Funktion wird geprüft, ob der Browser das Objekt `layer` kennt. Ist das der Fall, erfolgt die Zuweisung des Wertes über das `layer`-Objekt. Diesem Zweig der Anweisung folgt Netscape 4.x. Kennt der Browser das Objekt dagegen nicht, dann wird im `else`-Zweig geprüft, ob er das Objekt `window.netscape` kennt. Da dieses Objekt nur einem Netscape-basierten Browser bekannt ist, werden der MS Internet Explorer und Opera ausgeschlossen. Anschließend erfolgt die Zuweisung der Eigenschaft an das Element entsprechend der DOM-Syntax. Beide Zweige der `if/-else`-Anweisung verwenden jedoch die gleichen Event-Eigenschaften.

#### **Lesen** *screenX, screenY*

Netscape-Syntax. Speichert die horizontalen Pixel (`layerX`) und die vertikalen Pixel (`layerY`) der Cursor-Position absolut gesehen zum Bildschirm, wenn z.B. Mausereignisse überwacht werden. Ein Beispiel:

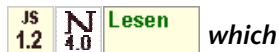
```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Position(Ereignis)
{
    alert("x-Wert: " + Ereignis.screenX + " / y-Wert: " + Ereignis.screenY);
}
document.onmousedown = Position;
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel überwacht, ob der Anwender die Maustaste drückt. Wenn ja, werden in einem Meldungsfester die absoluten Bildschirmkoordinaten ausgegeben, an denen die Maustaste gedrückt wurde.

**Beachten Sie:** Das vorliegende Beispiel wird auch vom Netscape 6 interpretiert.



Netscape-Syntax. Speichert bei Tastaturereignissen den dezimalen Code (ASCII/ANSI-Wert) der gedrückten Taste und bei Mausereignissen, welche Maustaste gedrückt wurde. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Position(Ereignis)
{
    alert("Maustaste: " + Ereignis.which);
}
document.onmousedown = Position;
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel überwacht, ob der Anwender die Maustaste drückt. Wenn ja, wird in einem Meldungsfester ausgegeben, welche Maustaste gedrückt wurde. Die linke Maustaste hat den Wert 1, die mittlere (sofern vorhanden) den Wert 2 und die rechte (sofern vorhanden) den Wert 3.

**Beachten Sie:** Das vorliegende Beispiel wird auch vom Netscape 6 interpretiert.



Netscape-Syntax. Speichert die Information, welches Ereignis eingetreten ist. Der gespeicherte Wert ist der Name des Events ohne das Präfix on, also beispielsweise mouseup oder keypress oder select. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Auswertung(Ereignis)
{
    alert("Ereignis: " + Ereignis.type);
}
document.onmouseup = Auswertung;
document.onkeyup = Auswertung;
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel überwacht zwei Ereignisse, nämlich, ob der Anwender die Maustaste drückt und ob er eine Taste drückt. In beiden Fällen wird die gleiche Handler-Funktion *Auswertung* aufgerufen. Sie gibt in einem Meldungsfester aus, welches der Ereignisse eingetreten ist.

**Beachten Sie:** Das vorliegende Beispiel wird auch vom Netscape 6 interpretiert.



Speichert die horizontalen Pixel ( $x$ ) und die vertikalen Pixel ( $y$ ) der Cursor-Position relativ zur oberen linken Ecke des Eltern-Elements von dem Element, das ein Ereignis ausgelöst hat. Wenn ein absolut positionierter Bereich das Eltern-Element ist, ist dessen obere linke Ecke der Bezugspunkt. Wenn das auslösende Element sonst kein Eltern-Element hat, gilt die linke obere Ecke des Dokuments als Bezug. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
function Coords()
{
    alert("x = " + window.event.x + " / y = " + window.event.y);
}
//-->
</script>
</head><body>
<div style="position:relative; left:50px; top:140px;
background-color:#FFFFE0; width:100px">
<p onClick="Coords()">Klick mich</p>
</div>
</body></html>
```

Das Beispiel enthält einen absolut positionierten Bereich (`<div>...</div>`) und innerhalb davon einen gewöhnlichen Textabsatz. Beim Anklicken des Absatzes wird die Funktion `Coords()` aufgerufen. Diese gibt die Pixelkoordinaten des Mausklicks aus. Eigentlich sollten sich die Werte dabei auf die linke obere Ecke des `<div>`-Bereichs beziehen.

**Beachten Sie:** Versionen des MS Internet Explorer vor der Version 5 speichern in diesen Eigenschaften die Position relativ zum Fenster. Ab dem MS Internet Explorer 5.0 ist in dieser Eigenschaft die Position zu einem relativ positionierten Elternelement (z.B. Tabellenzellen) gespeichert. Für absolut positionierte Elternelemente dagegen enthält die Eigenschaft wieder die Position relativ zum Fenster.

## 20.19 history

### Eigenschaften

`length` (Anzahl besuchter Seiten)

### Methoden

`back()` (zurückspringen)  
`forward()` (vorwärtsspringen)  
`go()` (zu URI in History springen)

**history: Allgemeines zur Verwendung**

Über das Objekt `history`, das in der JavaScript-Objekthierarchie unterhalb des `window`-Objekts liegt, haben Sie Zugriff auf die besuchten WWW-Seiten des Anwenders. Maßgeblich ist dabei die Liste, wie sie in der History-Liste des Web-Browsers gespeichert ist. Mit diesem Objekt können Sie z.B. Verweise vom Typ »springe zur zuletzt besuchten Seite« konstruieren. Einige Beispiele:

```
history.back();
window.history.back();
Zweitfenster.history.back();
parent.frames[2].history.back();
```

Methoden des `history`-Objekts können Sie mit `history` aufrufen, die zuletzt besuchte Seite beispielsweise mit `history.back()`. Sie können auch `window.history.back()` notieren. Da es sich dabei aber um den Inhalt des aktuellen Fensters handelt, ist die Angabe des Fensternamens nicht zwingend erforderlich. Anders beim Zugriff auf andere Fenster oder Frame-Fenster. In diesen Fällen müssen Sie das Fenster korrekt adressieren. Lesen Sie für zusätzliche Fenster den Abschnitt 20.2 zur *Verwendung des window-Objekts* und für Frame-Fenster den Abschnitt 20.3 über die *Verwendung des frames-Objekts*.

**Beachten Sie:** JavaScript erlaubt zum Schutz der Seitenbesucher nur einen begrenzten Zugriff auf das `history`-Objekt. Das bedeutet, Sie erhalten keinen Zugriff auf die URIs der besuchten Seitenadressen, und Sie können die History auch nicht löschen. Neben der Bewegung in der History ist es lediglich erlaubt, mit der Methode `location.replace()` den `history`-Eintrag der zuletzt besuchten Seite zu überschreiben.

   **Lesen** *length*

Speichert die Anzahl der Einträge in der History für ein Fenster. Das Beispiel definiert einen Verweis, bei dessen Anklicken die Anzahl der besuchten Seiten mit `alert()` ausgegeben wird.:

```
<a href="javascript:alert(history.length)">Anzahl besuchter Seiten;</a>
```

   **back()**

Lädt die WWW-Seite, die zuletzt besucht wurde. Erwartet keine Parameter. Das Beispiel definiert einen Verweis, bei dessen Anklicken die zuletzt besuchte Seite wieder aufgerufen wird.

```
<a href="javascript:history.back()">zur&uuml;ck</a>
```

   **forward()**

Lädt die WWW-Seite, die als nächste besucht wurde, sofern zuvor ein Back-Vorgang stattfand. Erwartet keine Parameter. Das Beispiel definiert einen Verweis, bei dessen Anklicken die Seite aufgerufen wird, die aktiv war, bevor der Anwender einen Back-Vorgang auslöste.

```
<a href="javascript:history.forward()">wieder vor</a>
```

   **go()**

Springt so viele Seiten in der gespeicherten History vor oder zurück wie gewünscht. Erwartet als Parameter die Anzahl der zu springenden Seiten. Ein negativer Wert springt rückwärts (so viele

Back-Vorgänge wie angegeben), ein positiver Wert springt vorwärts (so viele Forward-Vorgänge wie angegeben). Das Beispiel definiert einen Verweis, bei dessen Anklicken drei Seiten in der gespeicherten History der besuchten Seiten zurückgesprungen wird:

```
<a href="javascript:history.go(-3)">3 Seiten zur&uuml;ck</a>
```

## 20.20 location

### Eigenschaften

|          |                                       |
|----------|---------------------------------------|
| hash     | (Ankername innerhalb eines URI)       |
| host     | (Domain-Name innerhalb eines URI)     |
| hostname | (Domain-Name innerhalb eines URI)     |
| href     | (URI / Verweis zu URI)                |
| pathname | (Pfadname innerhalb eines URI)        |
| port     | (Portangabe innerhalb eines URI)      |
| protocol | (Protokollangabe innerhalb eines URI) |
| search   | (Parameter innerhalb eines URI)       |

### Methoden

|           |                                 |
|-----------|---------------------------------|
| reload()  | (neu laden)                     |
| replace() | (History-Eintrag überschreiben) |

### location: Allgemeines zur Verwendung

Über das Objekt `location`, das in der JavaScript-Objekthierarchie unterhalb des `window`-Objekts liegt, haben Sie Zugriff auf den vollständigen URI der aktuell angezeigten Web-Seite. Sie können den URI oder Teile davon zur Weiterverarbeitung abfragen und ändern. Beim Ändern führt der Web-Browser einen Sprung zu einem neuen URI aus, genauso wie bei einem Verweis. Einige Beispiele:

```
window.location.href = "datei2.htm";
MeineDomain = location.host;

parent.frames[1].location.href = "datei2.htm";
Zweitfenster.host = "www.teamone.de";
```

Methoden des `location`-Objekts können Sie mit `location` aufrufen. Sie können auch `window` davor notieren. Da es sich dabei aber um die Adresse des aktuellen Fensters handelt, ist die Angabe des Fensternamens nicht zwingend erforderlich. Anders beim Zugriff auf andere Fenster oder Frame-Fenster. In diesen Fällen müssen Sie das Fenster korrekt adressieren. Lesen Sie für zusätzliche Fenster den Abschnitt 20.2 zur *Verwendung des window-Objekts* und für Frame-Fenster den Abschnitt 20.3 über die *Verwendung des frames-Objekts*.



Speichert den Anker (Verweisziel) innerhalb des URI. Das Beispiel definiert zwei Verweise. Beim Anklicken des ersten Verweises wird zu dem Ziel `a2` innerhalb der Datei gesprungen. Dies ist

selbst wieder ein Verweis. Beim Anklicken dieses zweiten Verweises wird die Funktion `Zeige()` aufgerufen, die den Ankernamen ausgibt. Beachten Sie, dass das Gatterzeichen `#` mit zu dem Wert gehört, den `hash` liefert. Wenn Sie diese Angabe ändern, bewirken Sie einen Verweis innerhalb der Datei zu dem angegebenen Anker. Auch dabei müssen Sie das Gatterzeichen vor dem Ankernamen hinzufügen.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Zeige() { alert(window.location.hash); }
//-->
</script>
</head><body>
<a href="#a2">Gehe zu a2. Bitte zuerst klicken</a>
<p>
<a name="a2" href="javascript:Zeigen()">Hier a2 - bitte dann klicken</a>
</body></html>
```

**Beachten Sie:** Das vorliegende Beispiel wird von Opera 5.12 nicht interpretiert.



Speichert den Namen des Server-Rechners innerhalb des aktuellen oder fensterspezifischen URI. Das Beispiel gibt beim Einlesen der Datei aus, auf welchem Host-Rechner sich der Anwender befindet. Wenn es eine Namensadresse wie etwa *www.teamone.de* ist, wird diese ausgegeben. Wenn es eine numerische IP-Adresse wie etwa 195.201.34.18 ist, wird diese ausgegeben. Sie können die Eigenschaft ändern. Sicherer ist es in diesem Fall jedoch, mit der Eigenschaft `href` zu arbeiten.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert("Sie befinden sich auf dem Host " + window.location.host);
//-->
</script>
</head><body>
</body></html>
```



Speichert den Namen des Server-Rechners innerhalb des aktuellen oder fensterspezifischen URI, im Unterschied zu `host` auch Sub-Domains und Portnummern. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert("Sie befinden sich auf dem Host " + window.location.hostname);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel gibt beim Einlesen der Datei aus, auf welchem Host-Rechner sich der Anwender befindet. Wenn es eine Namensadresse wie etwa *www.teamone.de* ist, wird diese ausgegeben. Wenn es eine numerische IP-Adresse wie etwa 195.201.34.18 ist, wird diese ausgegeben. Wenn beispielsweise eine Portnummer dazugehört, wie *www.xy.de:5080*, wird auch diese gespeichert. Sie können die Eigenschaft ändern. Sicherer ist es in diesem Fall jedoch, mit der Eigenschaft *href* zu arbeiten.



Speichert den vollständigen URI des aktuellen oder eines anderen Fensters. Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird die Funktion *Wunsch()* aufgerufen. Diese Funktion blendet ein Dialogfenster ein, in dem der Anwender einen URI seiner Wahl eingeben kann. Der eingegebene Wert wird in der Variablen *Ziel* gespeichert. Anschließend wird durch Zuweisung von *Ziel* an *window.location.href* ein Verweis zu diesem URI ausgeführt. Es muss sich dabei nicht um einen vollständigen URI handeln. Ein Dateiname genügt, wenn der Sprung einfach nur zu einer anderen Projektdatei im gleichen Verzeichnis führen soll.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Wunsch() {
  var Ziel = window.prompt("Ihr Wunsch-URI:", "");
  window.location.href = Ziel;
}
//-->
</script>
</head><body>
<a href="javascript:Wunsch()">Wunschverweis</a>
</body></html>
```



Speichert den Pfadnamen innerhalb des aktuellen oder fensterspezifischen URI. Bei lokaler Verwendung (ohne HTTP-Protokoll) wird die vollständige lokale Adresse gespeichert. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert("Aktueller Pfad: " + window.location.pathname);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel gibt beim Einlesen der Datei in einem Meldungsfenster den Pfadnamen des URI der Datei aus. Sie können die Eigenschaft ändern. Sicherer ist es in diesem Fall jedoch, mit der Eigenschaft *href* zu arbeiten.



Speichert den Port innerhalb des aktuellen oder fensterspezifischen URI. Eine Adresse mit Ports ist etwa `http://www.xy.de:5080/` – die Eigenschaft `port` enthält nur dann einen Wert, wenn im aktuellen URI tatsächlich eine Portangabe enthalten ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(window.location.port != "")
  alert(window.location.port);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel ermittelt beim Einlesen der Datei, ob eine Portangabe vorhanden ist. Wenn ja, wird sie in einem Meldungsfenster ausgegeben. Sie können die Eigenschaft ändern. Dies kann jedoch zu Fehlermeldungen führen, wenn der Server den Port nicht kennt. Der »Normal-Port« für HTTP-Adressen ist 80.



Speichert das verwendete Protokoll innerhalb des aktuellen oder fensterspezifischen URI. Bei HTTP-Adressen wird `http:` gespeichert. Bei lokaler Verwendung (ohne HTTP-Protokoll) wird `file:` gespeichert. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert(window.location.protocol);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel gibt beim Einlesen der Datei das Protokoll aus, das zum aktuellen URI gehört. Sie können die Eigenschaft ändern. Dies kann jedoch zu Fehlermeldungen führen, wenn das angegebene Protokoll mit der gleichen sonstigen Adresse nicht unterstützt wird.



Speichert eine Zeichenkette, die, durch ein Fragezeichen getrennt, zum aktuellen URI gehört, zum Beispiel: `http://www.xy.de/index.htm?Browser=Netscape`. In diesem Fall enthält die Eigenschaft den Wert `?Browser=Netscape`. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(window.location.search != "")
  alert(window.location.search);
-->
```



```
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel ermittelt beim Einlesen der Datei, ob der URI einen Teil hinter einem Fragezeichen enthält. Wenn ja, wird sie in einem Meldungsfenster ausgegeben. Sie können (funktioniert nur mit Netscape) mit `href` eine andere Datei aufrufen und an den URI, durch ein Fragezeichen getrennt, Informationen anhängen, zum Beispiel solche, die Sie mit JavaScript ermittelt haben – Formulareingaben oder dergleichen. Die aufgerufene Datei kann solche Informationen auslesen und weiterverarbeiten. Auf diese Weise ist »Kommunikation« zwischen Dateien möglich.

**Beachten Sie:** Opera 5.12 zeigt im Protokoll `file:` den HTML-Quelltext einer Datei an, sobald diese einen Parameter erhält.



**reload()**

Lädt eine WWW-Seite neu. Hat den gleichen Effekt wie der Reload-Button im Browser. Das Beispiel definiert einen Verweis, bei dessen Anklicken die aktuelle Seite noch einmal neu geladen wird:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:location.reload()">Reload</a>
</body></html>
```



**replace()**

Lädt einen anderen URI über den aktuellen Eintrag in der Liste der besuchten Seiten (History). Hat dadurch einen ähnlichen Effekt wie die Eigenschaft `href`. Der Unterschied ist, dass der aktuelle URI anschließend in der Liste der besuchten Seiten nicht mehr vorkommt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<a href="javascript:location.replace('datei2.htm')">Datei 2</a>
</body></html>
```

Das Beispiel definiert einen Verweis, bei dessen Anklicken *datei2.htm* geladen wird. Der aktuelle URI verschwindet dabei aus der Liste der besuchten Seiten.

## 20.21 Array

### Eigenschaften

Assoziative Arrays  
`length`

(Arrays mit Zeichenketten als Schlüssel)  
(Anzahl Elemente)

**Methoden**

|                        |                                     |
|------------------------|-------------------------------------|
| <code>concat()</code>  | (Arrays verketten)                  |
| <code>join()</code>    | (Array in Zeichenkette umwandeln)   |
| <code>pop()</code>     | (letztes Array-Element löschen)     |
| <code>push()</code>    | (neue Array-Elemente anhängen)      |
| <code>reverse()</code> | (Elementreihenfolge umkehren)       |
| <code>shift()</code>   | (Erstes Array-Element entfernen)    |
| <code>slice()</code>   | (Teil-Array extrahieren)            |
| <code>splice()</code>  | (Elemente löschen und hinzufügen)   |
| <code>sort()</code>    | (Array sortieren)                   |
| <code>unshift()</code> | (Elemente am Array-Anfang einfügen) |

**Array: Allgemeines zur Verwendung**

Das Objekt `Array` ist als »Container« für Ketten gleichartiger Variablen gedacht. In der Programmiersprache spricht man auch von einem »Vektor«. Wenn Sie beispielsweise die 16 Grundfarben speichern wollen, brauchen Sie keine 16 Variablen, sondern ein `Array`-Objekt, in dem Sie 16 gleichartige Werte (im Beispiel: Farbwerte) speichern können.

Schema 1 / Beispiel 1:

```
Objektnamen = new Array();
MeineFrauen = new Array();
```

Schema 2 / Beispiel 2:

```
Objektnamen = new Array(Zahl);
MeineFrauen = new Array(100);
```

Schema 3 / Beispiel 3:

```
Objektnamen = new Array(Element0, Element1, ..., element_n);
MeineFrauen = new Array("Anita", "Bettina", "Christa", "Doris");
```

Eine Objektinstanz von `Array` speichern Sie in einem selbst vergebenen Objektnamen. Hinter dem Namen folgt ein Gleichheitszeichen. Dahinter folgt das reservierte Wort `new` und der Aufruf der Objektfunktion `Array()`.

Benutzen Sie Schema 1, wenn Sie zum Zeitpunkt der Definition noch nicht wissen, wie viele Elemente in dem Variablenvektor gespeichert werden sollen. Benutzen Sie Schema 2, wenn Sie zum Zeitpunkt der Definition bereits wissen, wie viele Elemente in dem Variablenvektor gespeichert werden sollen. Die Anzahl können Sie der Objektfunktion als Parameter übergeben. Benutzen Sie Schema 3, um den Variablenvektor gleich mit Anfangswerten vorzubelegen. Bei den Varianten 1 und 2 bleiben die einzelnen Variablen des Variablenvektors leer, bis ihnen im Programmverlauf ein Wert zugewiesen wird. Nachdem Sie eine Instanz des `Array`-Objekts erzeugt haben, können Sie es in Ihrem JavaScript-Code verwenden.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
Zahlen = new Array(34,86,167,413);
```

```
var x = Zahlen[1];
alert(x);
//-->
</script>
</head><body>
</body></html>
```

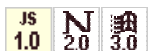
Nachdem Sie eine Instanz des Array-Objekts in einem Namen gespeichert haben (im Beispiel der Name `Zahlen`), können Sie wie in der zweiten Zeile gezeigt einzelne Werte innerhalb des Variablenvektors ansprechen. Im Beispiel wird eine gewöhnliche Variable `x` definiert. In dieser Variablen wird der Wert der zweiten Variablen des `Zahlen`-Vektors gespeichert, also 86. Beachten Sie, dass der Zähler bei 0 beginnt, d.h. den ersten Wert im Vektor sprechen Sie im Beispiel mit `Zahlen[0]` an, den zweiten mit `Zahlen[1]` usw.

**Beachten Sie:** Sie können auch mehrdimensionale Arrays erzeugen.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var a = new Array(4)
for (var i=0; i < a.length; ++i)
  a[i] = new Array(10);
a[3][1] = "Hallo";
alert(a[3][1]);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert zunächst einen Array `a` mit 4 Elementen. Dann wird in einer `for`-Schleife für jedes dieser Elemente ein neuer Array definiert, wobei für jeden Array 10 leere Elemente erzeugt werden. Anschließend können Sie durch eine Angabe wie `a[3][1]` das zweite Element (1) im vierten Array (3) ansprechen. Im Beispiel wird dieses Element mit dem Wert `Hallo` belegt. Zur Kontrolle wird der Wert in einem Meldungsfenster ausgegeben.



### Assoziative Arrays

Assoziative Arrays sind solche, bei denen der Zugriff auf einzelne Elemente mit Hilfe einer Zeichenkette erfolgt. Die Zeichenkette wird als **Schlüssel** für den Zugriff bezeichnet. Alle JavaScript-Objekte, die eine Eigenschaft `name` oder `id` besitzen, wie zum Beispiel die Objekte `forms` oder `elements`, stellen aus Sicht von JavaScript assoziative Arrays dar.

Assoziative Arrays sind besonders in Verbindung mit mehrdimensionalen Arrays sinnvoll. Sie ermöglichen es unabhängig von der Position, den Wert eines Elements zu ermitteln. Nachteil ist, dass die Methoden des Array-Objekts nicht anwendbar sind. Auch kennen assoziative Arrays keine Eigenschaft `length`. Elemente eines assoziativen Arrays können stets nur mit dem Namen oder über eine `for-in`-Schleife angesprochen werden. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
```

```

<!--
var Mitarbeiter=new Array();

Mitarbeiter[0] = new Array();
Mitarbeiter[0]["Name"] = "Müller";
Mitarbeiter[0]["Vorname"] = "Hans";
Mitarbeiter[0]["Wohnort"] = "Dresden";

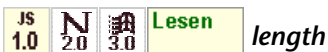
Mitarbeiter[1]=new Array()
Mitarbeiter[1]["Name"] = "Schulze";
Mitarbeiter[1]["Vorname"] = "Frauke";
Mitarbeiter[1]["Wohnort"] = "Berlin";

for (var i=0;i<Mitarbeiter.length;i++)
{
    document.write("<dl><dt>Mitarbeiter "+(i+1)+"</dt>");
    for (var Eigenschaft in Mitarbeiter[i])
        document.write("<dd>"+ Eigenschaft + ":
"+Mitarbeiter[i][Eigenschaft]+"</dd>");
    document.write("</dl>");
}
//-->
</script>
</body></html>

```

Das Beispiel definiert einen Array `Mitarbeiter`. Dieser Array besitzt zwei Elemente. Jedes dieser Elemente stellt selbst wieder einen Array dar. Dabei handelt es sich jedoch um assoziative Arrays, denn als Schlüssel werden die Begriffe `Name`, `Vorname` und `Wohnort` verwendet. Jedem dieser Elemente wird ein entsprechender Wert zugeordnet.

Die nachfolgende `for`-Schleife dient dazu, die Elemente des Arrays `Mitarbeiter` und die jeweiligen Eigenschaften auszugeben. Nach Beginn der `for`-Schleife wird zuerst mit der `write`-Methode der Beginn einer Definitionsliste in das Dokument geschrieben und dabei die laufende Nummer des Mitarbeiters angegeben. Dies entspricht der um 1 erhöhten Position im Array, da bei Array-Elementen die Zählung bei 0 beginnt. Anschließend wird eine weitere, innere Schleife aufgerufen. Da hierbei auf einen assoziativen Array zugegriffen wird, wird eine `for-in`-Schleife verwendet. Diese Schleife durchläuft jeden Index des angesprochenen assoziativen Arrays und schreibt dessen Eigenschaft und den zugehörigen Wert ins Dokument. Ist die `for-in`-Schleife beendet, so wird auch die Definitionsliste geschlossen und das nächste Element im Array `Mitarbeiter` durchlaufen.



**length**

Speichert die Anzahl der Elemente in einem Array. Beachten Sie, dass alle JavaScript-Objekte, die eine Eigenschaft `length` besitzen, wie zum Beispiel die Objekte `forms` oder `elements`, aus Sicht von JavaScript Arrays darstellen. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(1,2,4,8,16,32,64,128,256);
alert(Zahlen.length + " Zahlen sind definiert");

```

```
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert einen Array, bestehend aus 9 Zahlen. Anschließend wird zur Kontrolle ausgegeben, wie viele Elemente der Array enthält.

### **concat()**

Hängt einen Array an einen anderen an. Erwartet als Parameter den Namen des anzuhängenden Arrays. Gibt den zusammengefügt Array als Array zurück. Sie können auch mehrere Arrays als Parameter an die Methode übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(1,2,3);
var AndereZahlen = new Array(10,20,30);
ZahlenGesamt = Zahlen.concat(AndereZahlen);
alert(ZahlenGesamt.length);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert zunächst zwei Arrays `Zahlen` und `AndereZahlen` mit je drei Zahlen. In der Variablen `ZahlenGesamt` wird anschließend der Rückgabewert von `concat()` gespeichert, wobei die Methode auf den ersten definierten Array `Zahlen` angewendet wird und den zweiten Array `AndereZahlen` an diesen Array anhängt. Zur Kontrolle wird nach der Operation die Gesamtzahl der Elemente von `ZahlenGesamt` ausgegeben. Dieser Array enthält nun die Werte 1,2,3,10,20,30 und hat deshalb also 6 Elemente.

**Beachten Sie:** Die Methode `concat()` erzeugt unterschiedliche Referenzen auf die ihr übergebenen Arrays. Ist in den miteinander zu verknüpfenden Arrays ein Objekt gespeichert, so ist in dem neu erzeugten Array eine Art Zeiger auf dieses Objekt gespeichert. Das bedeutet: Wird das Objekt geändert, so ändert sich auch das von der Methode `concat()` erzeugte Array. Sind in den verknüpften Arrays jedoch Zahlen und Zeichenketten enthalten, so wird von diesen Werten eine Kopie erzeugt. Nachträgliche Änderungen haben dann keinen Einfluss auf das von der Methode `concat()` erzeugte Array. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen=new Array(1,2);
var mehrZahlen = new Array(Zahlen,3,4);
var AndereZahlen = new Array(5,6,7);

ZahlenGesamt = mehrZahlen.concat(AndereZahlen);
alert(ZahlenGesamt);

Zahlen[0]=70;
```

```

alert("Zahlen: " +Zahlen);

AndereZahlen[0]=70;
alert("AndereZahlen: " +AndereZahlen);

alert("ZahlenGesamt: " +ZahlenGesamt);
//-->
</script>
</head><body>
</body></html>

```

Im Beispiel wurden drei Arrays angelegt. Das erste Element im Array `mehrZahlen` ist dabei der Array `Zahlen`. Damit ist im Array `mehrZahlen` ein Objekt vom Typ `Array` gespeichert. Mit der Methode `concat()` wird an den Array `mehrZahlen` der Array `AndereZahlen` angehängt. In der Variablen `ZahlenGesamt` ist dann ein Array mit den Elementen 1,2,3,4,5,6,7 gespeichert.

Im nachfolgenden Script-Teil erhalten die ersten Elemente der Arrays `Zahlen` und `AndereZahlen` jeweils den Wert 70. Beide Arrays werden auch richtig geändert, wie die Kontrollausgabe mittels eines Meldungsfensters zeigt. Im von der Methode `concat()` erzeugten Array `ZahlenGesamt` sind jetzt die Werte 70,2,3,4,5,6,7 gespeichert. Das letzte Meldungsfenster zeigt dies an. Die Änderung des Arrays `Zahlen` wurde also übernommen, die Änderung im Array `AndereZahlen` dagegen nicht. Der Grund ist, dass der Array `Zahlen` als Objekt im Array `mehrZahlen` enthalten war und deshalb jede Änderung übernommen wird. Die Werte des Arrays `AndereZahlen` waren dagegen vom Typ `Zahl`, und es wurde lediglich eine Kopie angelegt. Nachträgliche Änderungen bleiben damit wirkungslos.

### `join()`

Verwandelt einen Array in eine Zeichenkette. Erwartet als Parameter ein oder mehrere Trennzeichen, durch das/die die Array-Einträge in der Zeichenkette voneinander getrennt sein sollen. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var Zahlen = new Array(1,2,4,8,16,32,64,128,256);
  var Zahlenkette = Zahlen.join(" ");
  alert(Zahlenkette);
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel definiert einen Array mit 9 Zahlen. Dann wird die Methode `join` auf die Variable `Zahlen` angewendet, in der der Array gespeichert ist. Dabei wird im Beispiel ein Leerzeichen als Parameter übergeben. Der Rückgabewert der Prozedur wird in der Variablen `Zahlenkette` gespeichert. Der Effekt ist, dass in `Zahlenkette` alle definierten Zahlen stehen, und zwar durch Leerzeichen voneinander getrennt. Zur Kontrolle wird das Ergebnis als Meldungsfenster ausgegeben.

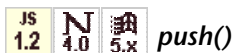


Entfernt das letzte Element aus einem Array und ändert dadurch die Anzahl der Elemente im Array. Der Rückgabewert der Methode ist der Inhalt des gelöschten Elementes. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(1,2,3,4,5);
alert(Zahlen.length + " Zahlen im Array");
function Entfernen()
{
    Zahlen.pop();
    alert(Zahlen.length + " Zahlen im Array");
}
//-->
</script>
</head><body>
<a href="javascript:Entfernen()">Zahl entfernen</a>
</body></html>
```

Das Beispiel definiert beim Einlesen der Datei einen Array *Zahlen* mit 5 Elementen. Zur Kontrolle wird in einem Meldungsfenster die Elementanzahl ausgegeben. Im Dateikörper enthält das Beispiel einen Verweis. Bei jedem Anklicken des Verweises wird die Funktion *Entfernen()* aufgerufen, die mit der *pop()*-Methode jeweils das letzte Element aus dem *Zahlen*-Array entfernt. Zur Kontrolle wird die neue Länge des Arrays ausgegeben.

**Beachten Sie:** Bei Browsern, die *pop()* nicht kennen, können Sie das letzte Element entfernen, indem Sie dem Array eine neue Länge zuweisen, z.B. mit *Zahlen.length=Zahlen.length-1*.



Hängt ein oder mehrere Elemente an das Ende eines Arrays an. Erwartet als Parameter ein oder mehrere anzuhängende Elemente. Gibt in der JavaScript-Version 1.2 das letzte (neueste) Element zurück, in neueren JavaScript-Versionen dagegen die Länge des Arrays. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Orte = new Array("Augsburg");
document.write(Orte.join(",") + '<p>');
var NeuesElement = Orte.push("Berlin");
document.write(Orte.join(",") + '<br>');
document.write('Neu dabei: ' + NeuesElement + '<p>');
var NeuesElement = Orte.push("Chemnitz","Dortmund","Essen","Frankfurt");
document.write(Orte.join(",") + '<br>');
document.write('Neuestes Element: ' + NeuesElement);
//-->
</script>
</body></html>
```

Das Beispiel erzeugt in einem JavaScript, das im Dateikörper steht, einen Array `Orte`, in dem zunächst nur ein Wert, nämlich `Augsburg`, gespeichert wird. Zur Kontrolle wird der Inhalt mit der `write()`-Methode in die Datei geschrieben. Anschließend wird dem Array mit der `push()`-Methode ein Element angehängt, nämlich `Berlin`. Der Rückgabewert der Operation wird in der Variablen `NeuestesElement` gespeichert. Zur Kontrolle wird dann der erweiterte Array und der Rückgabewert in die Datei geschrieben. Der Array besteht jetzt aus den Elementen `Augsburg` und `Berlin`. Das neueste Element ist `Berlin`. Dann werden nach dem gleichen Schema gleich vier neue Elemente an den Array angehängt. Der Array besteht hinterher aus den zwei bisherigen Städten plus den vier neuen. In der Variablen `NeuestesElement` wird jedoch nur das letzte, neueste Element gespeichert. Im Beispiel ist das `Frankfurt`. Zur Kontrolle werden auch diese Vorgänge in die Datei geschrieben.

**Beachten Sie:** Bei Browsern, welche die Methode `push()` nicht kennen, können Sie ein neues Element anhängen, indem Sie z.B. mit `Zahlen[Zahlen.length]="Wert"` das Array erweitern.



Keht die Elementreihenfolge innerhalb eines Arrays um. Das erste Element ist hinterher das letzte, das letzte das erste. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Name = new Array("S","T","E","F","A","N");
Name.reverse();
document.write(Name.join(""));
//-->
</script>
</body></html>
```

Das Beispiel definiert einen Array `Name`, dessen Elemente aus den einzelnen Buchstaben des Namens `Stefan` bestehen. Anschließend wird auf diesen Array die Methode `reverse()` angewendet. Mit Hilfe der `join()`-Methode wird der Array mit seiner neuen Elementreihenfolge anschließend so in die Datei geschrieben, dass keine Zeichen zwischen den Elementen stehen (das wird durch die leere Zeichenkette `" "` bei `join()` erreicht). Als Effekt erscheint der Name `Stefan` rückwärts geschrieben in der Datei.



Entfernt das erste Element aus einem Array. Die nachfolgenden Elemente rücken entsprechend nach vorne. Das bisher zweite Element wird das neue erste usw. Gibt das entfernte Element zurück. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(1,2,3,4,5);
alert("Erste Zahl im Array: " + Zahlen[0]);
function Entfernen()
{
```



```

var geloeschtesElement = Zahlen.shift();
alert("Erste Zahl im Array: " + Zahlen[0]);
}
//-->
</script>
</head><body>
<a href="javascript:Entfernen()">Zahl entfernen</a>
</body></html>

```

Das Beispiel definiert beim Einlesen der Datei einen Array *Zahlen* mit 5 Elementen. Zur Kontrolle wird in einem Meldungsfenster der Wert des ersten Elements ausgegeben, im Beispiel 1. Im Dateikörper enthält das Beispiel einen Verweis. Bei jedem Anklicken des Verweises wird die Funktion *Entfernen()* aufgerufen, die mit der *shift()*-Methode jeweils das erste Element aus dem *Zahlen*-Array entfernt. Zur Kontrolle wird das neue erste Element ausgegeben.



Extrahiert einen Teil aus einem Array. Erwartet als Parameter die Indexnummer des ersten und des letzten zu extrahierenden Elements. Als Indexnummer für das letzte Element kann auch ein negativer Wert übergeben werden. Dies bedeutet so viel wie »das soundsovielte Element von hinten«, wobei auch dort bei 0 zu zählen begonnen wird. Ein Wert von -1 bedeutet also das vorletzte Element. Die Methode gibt die extrahierten Elemente als neuen Array zurück. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(1,2,3,4,5);
for(var i = 0; i < Zahlen.length; ++i)
{
    var NeueZahlen = Zahlen.slice(i,Zahlen.length);
    alert(NeueZahlen.join(","));
}
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel definiert einen Array *Zahlen* mit 5 Elementen. Anschließend ist eine *for*-Schleife notiert. Die Schleife zählt von 0 bis 4. Innerhalb der Schleife wird bei jedem Schleifendurchgang das Ergebnis von *Zahlen.slice(i,Zahlen.length)* in einem neuen Array namens *NeueZahlen* gespeichert. Der Schleifenzähler *i* ist dabei die Indexnummer des ersten zu extrahierenden Elements. Das letzte zu extrahierende Element wird im Beispiel durch *Zahlen.length* angegeben (was allerdings eigentlich nicht ganz korrekt ist, da die Anzahl der Elemente 1 höher ist als der Index des letzten Elements – Netscape interpretiert es jedoch auf diese Weise korrekt, der MS Internet Explorer 4 interpretiert es dagegen eher wie erwartet. Dort muss man mit *Zahlen.length-1* arbeiten). Durch die Angaben werden alle Elemente von Element *i* bis zum letzten Element extrahiert. Da *i* bei jedem Schleifendurchgang höher wird, werden im Beispiel immer weniger Zahlen extrahiert, und zwar immer die hinteren des Arrays.

**Beachten Sie:** Die Methode *slice()* erzeugt unterschiedliche Referenzen auf die ihr übergebenen Arrays. Ist in den miteinander zu verknüpfenden Arrays ein Objekt gespeichert, so ist in

dem neu erzeugten Array eine Art Zeiger auf dieses Objekt gespeichert. Das bedeutet: Wird das Objekt geändert, so ändert sich auch der von der Methode `slice()` erzeugte Array. Sind in den verknüpften Arrays dagegen Zahlen und Zeichenketten enthalten, so wird eine Kopie erzeugt. Nachträgliche Änderungen haben dann keinen Einfluss auf den von `slice()` erzeugten Array. In ihrer Referenzierung verhält sich die Methode `slice()` genauso wie die Methode `concat()`.



Fügt ein oder mehrere neue Elemente in einen Array ein, und zwar an einer bestimmten gewünschten Stelle. Überschreibt dabei vorhandene Elemente. Erwartet folgende Parameter:

- *Startindex* = Indexnummer im Array, an der das oder die neuen Elemente eingefügt werden sollen.
- *Anzahl* = Anzahl der einzufügenden Elemente.
- ... *Element[e]* = neue Elemente.

Der 3. und die folgenden Parameter sind optional. Wenn Sie nur die ersten beiden Parameter angeben, werden leere Elemente eingefügt. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(1,2,3);
Zahlen.splice(1,3,4,5,6);
alert(Zahlen.join(","));
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert einen Array *Zahlen*, bestehend aus 5 Elementen mit den Zahlen 1 bis 3. Anschließend wird auf diesen Array die `splice()`-Methode angewendet, und zwar so: An Indexnummer 1, also bei der 2. Zahl im Array, werden 3 neue Elemente eingefügt, nämlich die Elemente mit den Werten 4, 5 und 6. Die ehemaligen Elemente mit den Werten 2 und 3 werden dabei mit den Werten 4 und 5 überschrieben. Das Element mit dem Wert 6 kommt neu hinzu. Hinterher enthält der Array also 4 Elemente mit folgenden Werten: 1,4,5,6.



Sortiert die Elemente eines Arrays. Wenn Sie keinen Parameter übergeben, wird lexikalisch sortiert, numerische Werte werden also intern in Zeichenketten verwandelt und wie Zeichenketten sortiert. Dabei kommt beispielsweise die Zahl 13 vor der Zahl 7, weil »13« als Zeichenkette lexikalisch vor »7« kommt. Wenn Sie numerische Werte sortieren wollen, können Sie eine Vergleichsfunktion definieren und deren Namen als Parameter übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
function Numsort(a,b)
{ return a-b; }
```

```

var Namen = new Array("Ina","Bettina","Tina","Martina");
Namen.sort();

var Zahlen = new Array(27,2,10,4);
Zahlen.sort(Numsort);

var Namensausgabe = Namen.join(",");
var Zahlenausgabe = Zahlen.join(",");
document.write("sortierte Namen: " + Namensausgabe + "<br>");
document.write("sortierte Zahlen: " + Zahlenausgabe);
//-->
</script>
</body></html>

```

Das Beispiel zeigt, wie Sie Zeichenketten und numerische Werte sortieren können. Im Beispiel wird ein Array `Namen` definiert, dessen Elemente Zeichenketten darstellen. Um einen solchen Array zu sortieren, genügt es, die Methode `sort()` einfach auf den Array anzuwenden. Die Elemente des Arrays werden dann neu angeordnet, nämlich lexikalisch aufsteigend sortiert. Um Zahlen zu sortieren, brauchen Sie eine einfache Vergleichsfunktion. Sie können dazu die im Beispiel definierte Funktion `Numsort()` verwenden. Das Beispiel definiert einen Array `Zahlen` und wendet die `sort()`-Methode auf diesen Array so an, dass der Funktionsname `Numsort` als Parameter übergeben wird. Dadurch werden die Elemente des Arrays numerisch sortiert.

Die Funktion `Numsort()` regelt, wie zwei Elemente sortiert werden. Sie besitzt die Parameter `a` und `b`. Während des Sortiervorganges wird die Funktion mit jeweils 2 Array-Elementen aufgerufen. Ein erwarteter Rückgabewert ist eine Zahl, die entweder kleiner, gleich oder größer als 0 ist. Eine solche Zahl entsteht durch die Differenz der beiden Parameter. Ist der Rückgabewert von `Numsort()` größer als 0, so bedeutet das, der Parameter `a` hat einen höheren Index als der Parameter `b`. In der Sortierreihenfolge kommt damit `b` vor `a`. Ist der Rückgabewert von `Numsort()` kleiner als 0, so bedeutet das, der Parameter `a` hat einen niedrigeren Index als der Parameter `b`. In der Sortierreihenfolge kommt damit `a` vor `b`. Tritt der Rückgabewert 0 ein, so sind beide Elemente gleich und brauchen in ihrer Reihenfolge nicht verändert werden.

Wenn Sie eine umgekehrte Sortierreihenfolge wünschen, wenden Sie nach der `sort()`-Methode die `reverse()`-Methode an.



Fügt am Anfang eines Arrays ein oder mehrere neue Elemente ein. Erwartet als Parameter die einzufügenden Elemente. Gibt die neue Elementzahl des Arrays zurück. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahlen = new Array(5,6,7,8,9);
var NeueAnzahl = Zahlen.unshift(1,2,3,4);
alert(NeueAnzahl);
alert(Zahlen.join(", "));
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel definiert einen Array `Zahlen` mit 5 Elementen, nämlich den Zahlen von 5 bis 9. Anschließend werden dem Array mit Hilfe der `unshift()`-Methode am Anfang 4 neue Zahlen hinzugefügt, nämlich die Zahlen von 1 bis 4. Der Rückgabewert der Operation wird in der Variablen `NeueAnzahl` gespeichert. Zur Kontrolle wird anschließend zunächst der Wert von `NeueAnzahl` ausgegeben und danach der komplette neue Array. Die Anzahl der Elemente ist nun 9, und der Array besteht aus den Elementen 1,2,3,4,5,6,7,8,9.

## 20.22 Boolean

### **Boolean: Allgemeines zur Verwendung**

Das Objekt `Boolean` ist zum Erzeugen von JavaScript-Standardwerten `true` (*wahr*) und `false` (*falsch*) gedacht. Solche Werte sind vor allem als Rückgabewerte für Funktionen gedacht. Boolesche Objekte werden bei der Definition immer mit einem der beiden möglichen Werte initialisiert und behalten diesen Wert auch. Die Variablen, in denen der Initialisierungswert gespeichert wird, stellen also Konstanten dar.

Beispiele für Initialisierung mit `true`:

```
wahr = new Boolean(true);
ja = new Boolean("irgendwas");
```

Beispiele für Initialisierung mit `false`:

```
falsch = new Boolean();
unwahr = new Boolean(0);
nix = new Boolean(null);
leer = new Boolean("");
Fehler = new Boolean(false);
```

Um eine Boolesche Konstante mit dem Wert `true` zu initialisieren, notieren Sie hinter dem frei vergebaren Namen der Konstante die Objektfunktion `new Boolean()`. Als Parameter übergeben Sie der Funktion entweder den Wert `true` (ohne Anführungszeichen) oder eine beliebige Zeichenkette.

Um die Boolesche Konstante mit dem Wert `false` zu initialisieren, übergeben Sie entweder gar nichts, oder den Wert `0`, oder den Wert `null` (ohne Anführungszeichen), oder den Wert `false` (ohne Anführungszeichen) oder eine leere Zeichenkette.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function LayerCheck() {
  var wahr = new Boolean(true);
  var falsch = new Boolean(false);
  if(document.layers) return wahr;
  else return falsch;
}
//-->
```

```

</script>
</head><body>
<a href="javascript:alert(LayerCheck())">auf Layer checken</a>
</body></html>

```

Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird der Rückgabewert der Funktion `LayerCheck()` ausgegeben, die in einem Script-Bereich im Dateikopf notiert ist. Wenn der Browser das `layers`-Objekt kennt, wird die zuvor definierte Konstante `wahr` zurückgegeben, wenn nicht, dann die Konstante `falsch`. Der Rückgabewert ist jedoch nicht der Name der Konstante, sondern der voreingestellte Wert in JavaScript. Es wird also entweder `true` oder `false` ausgegeben, je nachdem, ob der Browser Layer kennt oder nicht.

## 20.23 Date

### Methoden

|                                   |                                               |
|-----------------------------------|-----------------------------------------------|
| <code>getDate()</code>            | (Monatstag ermitteln)                         |
| <code>getDay()</code>             | (Wochentag ermitteln)                         |
| <code>getFullYear()</code>        | (volles Jahr ermitteln)                       |
| <code>getHours()</code>           | (Stundenteil der Uhrzeit ermitteln)           |
| <code>getMilliseconds()</code>    | (Millisekunden ermitteln)                     |
| <code>getMinutes()</code>         | (Minutenteil der Uhrzeit ermitteln)           |
| <code>getMonth()</code>           | (Monat ermitteln)                             |
| <code>getSeconds()</code>         | (Sekundenteil der Uhrzeit ermitteln)          |
| <code>getTime()</code>            | (Zeitpunkt ermitteln)                         |
| <code>getTimezoneOffset()</code>  | (Zeitzoneabweichung der Lokalzeit ermitteln)  |
| <code>getUTCDate()</code>         | (Monatstag von UTC-Zeit ermitteln)            |
| <code>getUTCDay()</code>          | (Wochentag von UTC-Zeit ermitteln)            |
| <code>getUTCFullYear()</code>     | (volles Jahr von UTC-Zeit ermitteln)          |
| <code>getUTCHours()</code>        | (Stundenteil der UTC-Uhrzeit ermitteln)       |
| <code>getUTCMilliseconds()</code> | (Millisekundenteil der UTC-Uhrzeit ermitteln) |
| <code>getUTCMinutes()</code>      | (Minutenteil der UTC-Uhrzeit ermitteln)       |
| <code>getUTCMonth()</code>        | (Monat von UTC-Uhrzeit ermitteln)             |
| <code>getUTCSeconds()</code>      | (Sekundenteil der UTC-Uhrzeit ermitteln)      |
| <code>getYear()</code>            | (Jahr ermitteln)                              |
| <code>parse()</code>              | (Millisekunden seit dem 1.1.1970 ermitteln)   |
| <code>setDate()</code>            | (Monatstag setzen)                            |
| <code>setFullYear()</code>        | (volles Jahr setzen)                          |
| <code>setHours()</code>           | (Stunden der Uhrzeit setzen)                  |
| <code>setMilliseconds()</code>    | (Millisekunden setzen)                        |
| <code>setMinutes()</code>         | (Minuten der Uhrzeit setzen)                  |
| <code>setMonth()</code>           | (Monat setzen)                                |
| <code>setSeconds()</code>         | (Sekunden der Uhrzeit setzen)                 |
| <code>setTime()</code>            | (Datum und Uhrzeit setzen)                    |
| <code>setUTCDate()</code>         | (Monatstag für UTC-Zeit setzen)               |
| <code>setUTCDay()</code>          | (Wochentag für UTC-Zeit setzen)               |
| <code>setUTCFullYear()</code>     | (volles Jahr für UTC-Zeit setzen)             |
| <code>setUTCHours()</code>        | (Stunden der UTC-Zeit setzen)                 |

|                                   |                                            |
|-----------------------------------|--------------------------------------------|
| <code>setUTCMilliseconds()</code> | (Millisekunden der UTC-Zeit setzen)        |
| <code>setUTCMinutes()</code>      | (Minuten der UTC-Zeit setzen)              |
| <code>setUTCMonth()</code>        | (Monat für UTC-Zeit setzen)                |
| <code>setUTCSeconds()</code>      | (Sekunden der UTC-Zeit setzen)             |
| <code>setYear()</code>            | (Datum und Uhrzeit setzen)                 |
| <code>toGMTString()</code>        | (Zeitpunkt in GMT-Format konvertieren)     |
| <code>toLocaleString()</code>     | (Zeitpunkt in lokales Format konvertieren) |
| <code>UTC()</code>                | (GMT-Zeit seit dem 1.1.1970 ermitteln)     |

### ***Date: Allgemeines zur Verwendung***

Das Objekt `Date` ist für alle Berechnungen mit Datum und Zeit zuständig. Dabei gibt es, wie in der EDV üblich, einen fixen historischen Zeitpunkt, der intern als Speicherungs- und Berechnungsbasis dient. In JavaScript ist dies – wie in der C- und Unix-Welt – der 1. Januar 1970, 0.00 Uhr. Die Einheit, in der in JavaScript intern Zeit berechnet wird, ist eine Millisekunde. Bevor Sie über JavaScript Zugriff auf die eingebauten Datums- und Uhrzeitfunktionen haben, müssen Sie ein neues `Date`-Objekt erzeugen. Dabei gibt es mehrere Varianten.

Variante 1:

```
Objektnamen = new Date();
```

Variante 2:

```
Objektnamen = new Date("Monat Tag, Jahr Stunden:Minuten:Sekunden");
```

Variante 3:

```
Objektnamen = new Date(Jahr, Monat, Tag);
```

Variante 4:

```
Objektnamen = new Date(Jahr, Monat, Tag, Stunden, Minuten, Sekunden);
```

Eine neues `Date`-Objekts speichern Sie in einem selbst vergebenen Objektnamen. Hinter dem Namen folgt ein Gleichheitszeichen. Dahinter folgt das reservierte Wort `new` und der Aufruf der Objektfunktion `Date()`.

Benutzen Sie die Variante 1, wenn Sie eine Objektinstanz erzeugen wollen, in der das zum Zeitpunkt der Programmausführung aktuelle Datum und die aktuelle Uhrzeit gespeichert werden sollen. Benutzen Sie eine der Varianten 2 bis 4, wenn Sie das neue `Date`-Objekt mit bestimmten Werten (also einem bestimmten Datum und einer bestimmten Uhrzeit) initialisieren wollen. Alle Initialisierungswerte wie Monat oder Minuten müssen Sie in Zahlenform angeben, also etwa 9 für Monat Oktober. Ausnahme: Bei Variante (2) übergeben Sie die Initialisierungsdaten als Zeichenkette. Dabei wird der Monat in englischer Schreibweise angegeben, also beispielsweise `october`. Wenn Sie einen Monat als Zahlenwert übergeben, so wie in den Varianten 3 und 4, müssen Sie bei 0 zu zählen beginnen. Für Januar müssen Sie also 0 übergeben, für Februar 1, und für Dezember 11. Dies ist auch der Grund, warum für den Monat Oktober nicht 10, sondern 9 übergeben wird. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function bis2020() {
```

```

var jetzt = new Date();
var Zeit = jetzt.getTime() / 1000;
var Jahr2020 = new Date(2020,0,1,0,0,0);
var Endzeit = Jahr2020.getTime() / 1000;
var Rest = Math.floor(Endzeit - Zeit);
alert("Noch " + Rest + " Sekunden bis zum Jahr 2020");
}
//-->
</script>
</head><body>
<a href="javascript:bis2020()">Wie lange noch?</a>
</body></html>

```

Das Beispiel zeigt eine vollständige JavaScript-Funktion. Die Funktion ermittelt die Anzahl Sekunden zwischen aktuellem Zeitpunkt und dem Jahr 2020. Dazu wird in `jetzt` eine neue Instanz von `Date` mit der aktuellen Zeit erzeugt. Für den Zeitpunkt, an dem das Jahr 2020 beginnt, wird eine zweite Objektinstanz von `Date` erzeugt, nämlich in `Jahr2020`. Dazu werden der Objektfunktion `Date()` entsprechende Parameter übergeben. Über die beiden Objektnamen `jetzt` bzw. `Jahr2020` wird dann die Methode `Objektname.getTime()` aufgerufen. `getTime()` liefert die Anzahl Millisekunden seit dem 1.1.1970 zurück. In der Variablen `Zeit` wird auf diese Weise die Anzahl Sekunden zwischen dem 1.1.1970 und dem aktuellen Zeitpunkt gespeichert, und in der Variablen `Endzeit` die Anzahl Sekunden zwischen dem 1.1.1970 und dem 1.1.2020. Die Differenz der beiden Werte wird in der Variablen `Rest` gespeichert und mit `alert()` in einem Meldungsfenster am Bildschirm ausgegeben.

**Beachten Sie:** Alle Methoden, mit denen Sie einen aktuellen Zeitpunkt oder einen Teil davon, etwa den Wochentag oder den Monat, ermitteln können, beziehen sich auf die Zeit, die auf der Systemzeit des Anwenderrechners basiert. Denn JavaScript wird beim Anwender von dessen Browser ausgeführt. Die aktuelle Systemzeit des Server-Rechners können Sie mit JavaScript nicht ermitteln – das geht beispielsweise mit einem CGI-Script.



Liefert den Monatstag des Objekts `Objektname` als Zahl zurück, beispielsweise 2, wenn in `Objektname` der 2.10.1991, 23:59:00 gespeichert ist. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var jetzt = new Date();
var Tag = jetzt.getDate();
document.write('<table border bgcolor="#CCFFFF"><tr><td><font size="7">');
document.write(Tag + "</font></td></tr></table>");
//-->
</script>
</body></html>

```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getDate()` wird das aktuelle Tagesdatum innerhalb des Monats ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `Tag` gespeichert. Anschließend wird mit

Hilfe von `document.write()` dynamisch eine kleine Tabelle in die Datei geschrieben, in der das aktuelle Tagesdatum formatiert ausgegeben wird. Damit ein Maskieren der " entfällt, wurde als Textbegrenzerzeichen das ' gewählt.

### **getDay()**

Liefert den Wochentag des Objekts `Objektnamen` als Zahl zurück, und zwar in Form von Zahlen zwischen 0 (für Sonntag), 1 (für Montag) usw. bis 6 (für Samstag), also beispielsweise 3, wenn in `Objektnamen` der 2.10.1991, 23:59:00, gespeichert ist, da der 2.10.1991 ein Mittwoch war. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var jetzt = new Date();
var TagInWoche = jetzt.getDay();
var Wochentag = new
Array("Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag");
document.write('<table border bgcolor="#CCFFFF"><tr><td><b>');
document.write(Wochentag[TagInWoche] + "<\b></td></tr></table>");
//-->
</script>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getDay()` wird der numerische Wert des aktuellen Wochentags ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `TagInWoche` gespeichert. Es soll jedoch ein richtiger Wochentagsname ausgegeben werden. Deshalb wird im Beispiel ein `Array`-Objekt mit deutschen Wochentagsnamen erzeugt. Das erste Element ist dabei der Sonntag, da der Rückgabewert von `getDay()` ebenfalls intern mit Sonntag als erstem Wochentag rechnet. Anschließend wird mit Hilfe von `document.write()` dynamisch eine kleine Tabelle in die Datei geschrieben, in der das aktuelle Tagesdatum formatiert ausgegeben wird. Beachten Sie, dass die Variable `TagInWoche` dabei als Index für die Ausgabe des richtigen Elements aus dem `Array` `Wochentag` benutzt wird.

### **getFullYear()**

Liefert im Gegensatz zu `getYear()` beim Netscape-Browser das volle Jahr zurück. Beim Internet Explorer liefern beide Methoden den gleichen Wert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var jetzt = new Date();
var Jahr = jetzt.getFullYear();
document.write("Wir befinden uns im Jahr " + Jahr);
//-->
</script>
</body></html>
```



Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getFullYear()` wird das volle aktuelle Jahr ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `Jahr` gespeichert. Anschließend wird zur Kontrolle ein Text mit der Ausgabe dieser Variablen ins Dokumentfenster geschrieben.

### **`getHours()`**

Liefert die Stunden der Uhrzeit des Objekts `Objektname` als Zahl zurück, beispielsweise 23, wenn in `Objektname` der 2.10.1991, 23:59:00, gespeichert ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var jetzt = new Date();
var Std = jetzt.getHours();
if(Std >= 5 && Std < 12) alert("Guten Morgen!");
else if(Std >= 12 && Std < 18) alert("Guten Tag!");
else if(Std >= 18 && Std <= 23) alert("Guten Abend!");
else if(Std >= 0 && Std < 5) alert("Zeit, ins Bett zu gehen!");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getHours()` wird die Stunde des Tages ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `Std` gespeichert. Anschließend wird ein Meldungsfenster mit einem tageszeitabhängigen Text ausgegeben, je nachdem, in welchem Bereich der Wert von `Std` liegt.

### **`getMilliseconds()`**

Ermittelt die Anzahl der Millisekunden seit der letzten vollen Sekunde. Das Beispiel enthält ein Formular mit einem Eingabefeld und einem Klick-Button. Beim Anklicken des Buttons wird die Funktion `aktualisieren()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `zeit`. Mit `zeit.getMilliseconds()` werden die Millisekunden seit der letzten vollen Sekunde ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `ms` gespeichert. Anschließend wird der Wert des Formulareingabefeldes mit dem ermittelten Wert belegt, um das Ergebnis anzuzeigen.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function aktualisieren() {
var zeit = new Date();
var ms = zeit.getMilliseconds();
document.Formular.Ausgabe.value = ms;
}
//-->
</script>
</head><body onload="document.Formular.Ausgabe.value='0'">
<form name="Formular" action="">
```

```
<input name="Ausgabe" type="text">
<input type="button" value="aktualisieren" onClick="aktualisieren()">
</form>
</body></html>
```

### **getMinutes()**

Liefert die Minuten der Uhrzeit des Objekts `Objektnamen` als Zahl zurück, beispielsweise 59, wenn in `Objektnamen` der 2.10.1991, 23:59:00 gespeichert ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var jetzt = new Date();
  var Std = jetzt.getHours();
  var Min = jetzt.getMinutes();
  var StdAusgabe = ((Std < 10) ? "0" + Std : Std);
  var MinAusgabe = ((Min < 10) ? "0" + Min : Min);
  alert("Bei Ihnen ist es jetzt " + StdAusgabe + "." + MinAusgabe + " Uhr");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getHours()` wird die Stunde des Tages ermittelt, mit `jetzt.getMinutes()` die Minutenzahl. Der Rückgabewert der Aufrufe wird in den Variablen `Std` und `Min` gespeichert. Dann wird die Stundenzahl und die Minutenzahl für die Ausgabe formatiert. Wenn die Anzahl der Stunden oder Minuten kleiner 10 ist, wird eine 0 davorgesetzt. Schließlich wird ein Meldungsfenster mit der aktuellen Uhrzeit ausgegeben.

### **getMonth()**

Liefert den Monat des Objekts `Objektnamen` als Zahl zurück, beispielsweise 9, wenn in `Objektnamen` der 2.10.1991, 23:59:00, gespeichert ist. Beachten Sie, dass die Monatszählung bei 0 beginnt. 0 ist also Januar, 1 ist Februar usw. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var jetzt = new Date();
  var Jahresmonat = jetzt.getMonth();
  var Monat = new
  Array("Januar", "Februar", "M&auml;r", "April", "Mai", "Juni",
  "Juli", "August", "September", "Oktober", "November", "Dezember");
  document.write('<table border bgcolor="#CCFFFF"><tr><td><b>');
  document.write(Monat[Jahresmonat] + "<\b></td></tr></table>');
//-->
</script>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getMonth()` wird der numerische Wert des aktuellen Monats ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `Jahresmonat` gespeichert. Es soll jedoch ein richtiger Monatsname ausgegeben werden. Deshalb wird im Beispiel ein Array-Objekt mit deutschen Monatsnamen erzeugt.

Anschließend wird mit Hilfe von `document.write()` dynamisch eine kleine Tabelle in die Datei geschrieben, in der der aktuelle Monat formatiert ausgegeben wird. Beachten Sie, dass die Variable `Jahresmonat` dabei als Index für die Ausgabe des richtigen Elements aus dem Array `Monat` benutzt wird. Da auch bei Arrays bei 0 zu zählen begonnen wird, wird beispielsweise »Januar« geschrieben, wenn in `Jahresmonat` der Wert 0 gespeichert ist, oder »Februar«, wenn in `Jahresmonat` 1 gespeichert ist.



Liefert die Sekunden der Uhrzeit des Objekts `Objektname` als Zahl zurück, beispielsweise 0, wenn in `Objektname` der 2.10.1991, 23:59:00, gespeichert ist. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zaehler = 0;
function Anzeige() {
    var jetzt = new Date();
    var Sekunden = jetzt.getSeconds();
    alert(Sekunden);
    Zaehler = Zaehler + 1;
    if(Zaehler < 3) Anzeige();
}
Anzeige();
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert eine Funktion `Anzeige()`. Darin wird ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt` erzeugt. Mit `jetzt.getSeconds()` wird die Sekundenzahl innerhalb der aktuellen Minute der Uhrzeit ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `Sekunden` gespeichert. Dann wird die Sekundenzahl ausgegeben. Nachdem der Anwender das Meldungsfenster mit der Anzeige bestätigt hat, wird die Variable `Zaehler` um 1 erhöht. Solange diese Variable kleiner 3 ist, wird die Funktion `Anzeige()` erneut aufgerufen. Dabei werden alle Anweisungen innerhalb der Funktion wiederholt.



Liefert die Anzahl Millisekunden als Zahl zurück, die seit dem 1. Januar 1970, 0:00:00, bis zu dem in `Objektname` gespeicherten Zeitpunkt vergangen sind. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
```

```

var jetzt = new Date();
alert(jetzt.getTime());
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `alert(jetzt.getTime())` wird die aktuelle Anzahl von Millisekunden, die seit dem 1.1.1970 vergangen sind, in einem Meldungsfenster ausgegeben. Probieren Sie das ruhig mal aus, um ein Gefühl für die Größenordnung der Werte zu bekommen, die `getTime()` liefert.

### **`getTimezoneOffset()`**

Liefert den Unterschied zwischen lokaler Rechnerzeit und Greenwich Mean Time (GMT) in Anzahl Minuten zurück. Je nach lokaler Zeitzone, die in der Systemzeit des Anwenderrechners definiert ist, ist der Rückgabewert positiv oder negativ. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
var jetzt = new Date();
var Unterschied = jetzt.getTimezoneOffset() / 60;
alert("Unterschied zu Greenwich: " + Unterschied + " Stunde(n)");
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel möchte den Unterschied zwischen Greenwich-Zeit und lokaler Zeit in Stunden ausgeben. Dazu erzeugt es ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Dann wird die Methode `getTimezoneOffset()` für dieses Datumobjekt aufgerufen. Um keine Minutenzahlen, sondern Stundenzahlen zu erhalten, wird der Rückgabewert gleich durch 60 geteilt. Das Ergebnis wird in der Variablen `Unterschied` gespeichert. In einem Meldungsfenster wird das Ergebnis anschließend ausgegeben.

**Beachten Sie:** Die Sommerzeit wird von dieser Methode nicht berücksichtigt.

### **`getUTCDate()`**

Diese Methode ist funktional identisch mit `getDate()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.

### **`getUTCDay()`**

Diese Methode ist funktional identisch mit `getDay()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getUTCFullYear()***

Diese Methode ist funktional identisch mit `getFullYear()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getUTCHours()***

Diese Methode ist funktional identisch mit `getHours()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getUTCMilliseconds()***

Diese Methode ist funktional identisch mit `getMilliseconds()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getUTCMinutes()***

Diese Methode ist funktional identisch mit `getMinutes()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getUTCMonth()***

Diese Methode ist funktional identisch mit `getMonth()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getUTCSeconds()***

Diese Methode ist funktional identisch mit `getSeconds()`, liefert ihren Wert jedoch nicht für die lokale Zeit, sondern für die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




***getYear()***

Liefert das Jahr des Objekts `Objektname` als Zahl zurück, beispielsweise 1991, wenn in `Objektname` der 2.10.1991, 23:59:00, gespeichert ist. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var jetzt = new Date();
var Jahr = jetzt.getYear();
if(Jahr < 999) Jahr+=1900;
document.write("<h1>" + Jahr + "</h1>");
```

```
//-->
</script>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt mit dem aktuellen Zeitpunkt in dem Objektnamen `jetzt`. Mit `jetzt.getYear()` wird das aktuelle Jahr ermittelt. Der Rückgabewert des Aufrufs wird in der Variablen `Jahr` gespeichert. Manche Browser geben die Jahreszahl noch zweistellig aus. Für diesen Fall wird die Jahreszahl um 1900 erhöht. Anschließend wird mit Hilfe von `document.write()` dynamisch eine Zeile in die Datei geschrieben, in der das aktuelle Jahr formatiert ausgegeben wird.

### **parse()**

Ermittelt aus einer als Parameter zu übergebenden Zeichenkette `Zeitpunkt` die Anzahl Millisekunden, die zwischen dem 1.1.1970 0:00:00, und dem übergebenen Zeitpunkt verstrichen sind. Beachten Sie, dass Sie zum Aufrufen dieser Methode kein neues `Date`-Objekt erzeugen müssen, sondern direkt mit `Date` arbeiten können. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var DreissigJahre = Date.parse("Tue, 1 Jan 2000 00:00:00 GMT");
  alert(DreissigJahre);
-->
</script>
</head><body>
</body></html>
```

Das Beispiel ermittelt die Anzahl Millisekunden zwischen dem 1.1.1970 und dem 1.1.2000. Das Ergebnis wird in der Variablen `DreissigJahre` gespeichert. Zur Kontrolle wird die Zahl in einem Meldungsfenster ausgegeben.

### **setDate()**

Ändert den Monatstag, der in `Objektnamen` gespeichert ist. Der neue Monatstag ist als Parameter zu übergeben. Achten Sie darauf, dass Sie sinnvolle Werte zwischen 1 und 31 übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var Zeit = new Date("March 10, 1998 22:48:00");
  var Tag = Zeit.getDate();
  alert(Tag);
  Zeit.setDate(1);
  Tag = Zeit.getDate();
  alert(Tag);
-->
</script>
</head><body>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt `Zeit` mit einem bestimmten Zeitpunkt, der im GMT-Format angegeben wird. Zur Kontrolle ermittelt das Beispiel den aktuellen Tag im Monat aus dem definierten Zeitpunkt und gibt ihn in einem Meldungsfenster aus. Dann wird die Methode `setDate()` auf den Objektnamen `Zeit` angewendet. Im Beispiel wird der Tag des Monats auf 1 gesetzt. Anschließend wird zur Kontrolle der neue Tag im Monat ermittelt und anschließend ausgegeben.



### ***setFullYear()***

Ändert die Jahreszahl, die in `Objektnamen` gespeichert ist. Das neue Jahr ist als Parameter zu übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var zeit = new Date();
  var zukunft = zeit.setFullYear(2037);
  document.write("irgendwann schreiben wir das Jahr " + zeit.getUTCFullYear());
//-->
</script>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt `zeit`. Dann wird die Methode `setFullYear()` auf den Objektnamen `zeit` angewendet. Im Beispiel wird das Jahr auf 2037 gesetzt. Anschließend wird das Ergebnis zur Kontrolle ins Dokument geschrieben.



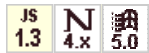
### ***setHours()***

Ändert die Stunden der Uhrzeit, die in `Objektnamen` gespeichert sind. Die Stundenzahl ist als Parameter zu übergeben. Achten Sie darauf, dass Sie sinnvolle Werte zwischen 0 und 23 übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var Zeit = new Date();
  var Std = Zeit.getHours();
  alert(Std);
  Zeit.setHours(23);
  Std = Zeit.getHours();
  alert(Std);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt `Zeit` mit dem aktuellen Zeitpunkt. Zur Kontrolle ermittelt das Beispiel die Stundenzahl aus der Tageszeit und gibt sie in einem Meldungsfenster aus. Dann wird die Methode `setHours()` auf den Objektnamen `Zeit` angewendet. Im Beispiel

wird die Stundenzahl der Uhrzeit auf 23 gesetzt. Anschließend wird zur Kontrolle der neue Stundenanteil an der Uhrzeit ermittelt und anschließend ausgegeben.

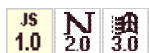


### **setMilliseconds()**

Ändert die Millisekunden des Zeitpunkts, der in `Objektname` gespeichert ist. Die Millisekundenzahl ist als Parameter zu übergeben. Achten Sie darauf, dass Sie sinnvolle Werte zwischen 0 und 999 übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function set0() {
    zeit = new Date();
    zeit.setMilliseconds(0);
}
function getNow() {
    zeit = new Date();
    alert(zeit.getMilliseconds() + " ms sind zwischen den Klicks vergangen");
}
//-->
</script>
</head><body>
<p>Klicken Sie auf den ersten Verweis und dann ganz schnell auf den zweiten!</p>
<p><a href="javascript:set0()">erster</a>
<a href="javascript:getNow()">zweiter</a></p>
</body></html>
```

Das Beispiel enthält zwei Verweise. Der Anwender soll zunächst den ersten anklicken und dann innerhalb einer Sekunde den zweiten. Beide Verweise rufen jeweils eine Funktion auf, die im Dateikopf notiert ist. Der erste Verweis ruft `set0()` auf. Darin wird ein neues Datumobjekt `zeit` mit dem aktuellen Zeitpunkt erzeugt. Anschließend wird mit `zeit.setMilliseconds(0)` die Anzahl der Millisekunden auf 0 gesetzt. Die Funktion `getNow()`, die der andere Verweis aufruft, erzeugt ebenfalls ein Datumobjekt mit dem aktuellen Zeitpunkt, gibt dann aber mit `zeit.getMilliseconds()` die verstrichenen Millisekunden seit dem Setzen auf 0 aus.



### **setMinutes()**

Ändert die Minuten der Uhrzeit, die in `Objektname` gespeichert sind. Die Minutenzahl ist als Parameter zu übergeben. Achten Sie darauf, dass Sie sinnvolle Werte zwischen 0 und 59 übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Andersrum() {
    var Zeit = new Date();
    var Min = Zeit.getMinutes();
    var MinNeu = Zeit.setMinutes(60-Min);
    var Min = Zeit.getMinutes();
}
```



```

return(Min);
}
//-->
</script>
</head><body>
<a href="javascript:alert(Andersrum())">Test</a>
</body></html>

```

Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird der Rückgabewert der Funktion `Andersrum()` ausgegeben, die in einem Script-Bereich im Dateikopf notiert ist. Dort wird ein neues Datumobjekt `Zeit` mit dem aktuellen Zeitpunkt erzeugt. Aus dem Zeitpunkt wird die Minutenzahl aus der Uhrzeit ermittelt. Mit der Methode `setMinutes()` wird die Minutenzahl innerhalb der Stunde umgekehrt. War es vorher beispielsweise 18.12 Uhr, ist es anschließend 18.48 Uhr. Der neue Minutenanteil an der Uhrzeit wird noch mal ermittelt und anschließend zurückgegeben. Eine solche Funktion könnte etwa als Teil einer rückwärts gehenden Uhr Sinn machen.

### **setMonth()**

Ändert den Monat, der in `Objektnamen` gespeichert ist. Der gewünschte Wert ist als Parameter zu übergeben. Achten Sie darauf, dass Sie sinnvolle Werte zwischen 0 und 11 übergeben. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
function andererMonat() {
    var Zeit = new Date();
    var Tag = Zeit.getDate();
    Zeit.setMonth(Math.round(Math.random() * 11));
    var Monat = Zeit.getMonth();
    return(Tag + "." + (Monat+1) + ".");
}
//-->
</script>
</head><body>
<a href="javascript:alert(andererMonat())">Test</a>
</body></html>

```

Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird der Rückgabewert der Funktion `andererMonat()` ausgegeben, die in einem Script-Bereich im Dateikopf notiert ist. Dort wird ein neues Datumobjekt `Zeit` mit dem aktuellen Zeitpunkt erzeugt. Mit der Methode `setMonth()` wird ein neuer Monat gesetzt – und zwar per Zufall. Dazu wird mit Hilfe von `Math.round()` und `Math.random()` eine zufällige Ganzzahl zwischen 0 und 11 ermittelt. Diese Zahl wird der `setMonth()`-Methode als Parameter übergeben. Die neue Monatszahl wird noch mal ermittelt und anschließend so formatiert zurückgegeben, dass ein sinnvolles Tagesdatum angezeigt wird.




**setSeconds()**

Ändert die Sekunden der Uhrzeit, die in `Objektname` gespeichert sind. Die Sekundenzahl ist als Parameter zu übergeben. Achten Sie darauf, dass Sie sinnvolle Werte zwischen 0 und 59 übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Uhrzeit() {
  var Zeit = new Date();
  var Std = Zeit.getHours();
  var Min = Zeit.getMinutes();
  var Sek = Zeit.getSeconds();
  var SekNeu = Zeit.setSeconds(Math.floor(Sek / 10));
  Sek = Zeit.getSeconds();
  return(Std + ":" + Min + ":" + Sek + "0");
}
//-->
</script>
</head><body>
<a href="javascript:alert(Uhrzeit())">Uhrzeit</a>
</body></html>
```

Das Beispiel enthält einen Verweis. Beim Anklicken des Verweises wird der Rückgabewert der Funktion `Uhrzeit()` ausgegeben, die in einem Script-Bereich im Dateikopf notiert ist. Das ist die aktuelle Uhrzeit, jedoch so ausgegeben, dass Sekunden immer nur in 10er-Schritten ausgegeben werden. Eine Ausgabe wie 18.10:37 ist also nicht möglich, die Funktion macht 18:10:30 daraus. Zu diesem Zweck wird in der Funktion ein neues Datumobjekt `Zeit` mit dem aktuellen Zeitpunkt erzeugt. Aus dem Zeitpunkt werden die Bestandteile der Uhrzeit ermittelt. Mit der Methode `setSeconds()` wird die Sekundenzahl behandelt. Durch eine kleine mathematische Operation wird aus der möglichen Sekundenzahl zwischen 0 und 59 eine Zahl zwischen 0 und 5. Der neue Sekundenanteil an der Uhrzeit wird noch mal ermittelt. Beim Zurückgeben der Uhrzeit wird an die einstellige Sekundenzahl eine 0 angehängt, denn »in Wirklichkeit« stellt die einstellige Zahl ja den Zehneranteil an den Sekunden dar.




**setTime()**

Ändert den kompletten Inhalt von `Objektname` auf einmal, erlaubt also das Setzen eines beliebigen neuen Zeitpunkts in `Objektname`. Erwartet als Parameter eine Zahl, die den neuen Zeitpunkt markiert. Die Zahl wird als Anzahl Millisekunden seit dem 1.1.1970 0:00:00 interpretiert. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zeit = new Date();
var AbsolutJetzt = Zeit.getTime();
var AbsolutDann = AbsolutJetzt + (100*24*60*60*1000);
Zeit.setTime(AbsolutDann);
```

```

var Jahr = Zeit.getYear();
var Monat = Zeit.getMonth() + 1;
var Tag = Zeit.getDate();
alert("in 100 Tagen ist der " + Tag + "." + Monat + "." + Jahr);
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel gibt aus, welches Datum von jetzt an in 100 Tagen aktuell sein wird. Zu diesem Zweck wird ein neues Datumobjekt *Zeit* mit dem aktuellen Zeitpunkt erzeugt. Mit *getTime()* wird zunächst die absolute Anzahl Millisekunden seit dem 1.1.1970 für den aktuellen Zeitpunkt ermittelt und in der Variablen *AbsolutJetzt* gespeichert. Anschließend wird zu dieser gespeicherten Zahl das Produkt aus  $100 \cdot 24 \cdot 60 \cdot 60 \cdot 1000$  hinzuaddiert. Das ist die Anzahl Millisekunden für 100 Tage. Das Ergebnis wird in der Variablen *AbsolutDann* gespeichert. Dieser Wert wird der Methode *setTime()* übergeben. Dadurch wird die Zeit auf einen Zeitpunkt in 100 Tagen gesetzt. Mit Hilfe entsprechender Befehle werden dann aus dem neu gesetzten Zeitpunkt die Werte für den Monatstag, den Monat und das Jahr ermittelt. Das Ergebnis wird in einem Meldungsfenster ausgegeben.

#### **setUTCDate()**

Diese Methode ist funktional identisch mit *setDate()*. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.

#### **setUTCDay()**

Diese Methode ist funktional identisch mit *setDay()*. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.

#### **setUTCFullYear()**

Diese Methode ist funktional identisch mit *setFullYear()*. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.

#### **setUTCHours()**

Diese Methode ist funktional identisch mit *setHours()*. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.

#### **setUTCMilliseconds()**

Diese Methode ist funktional identisch mit *setMilliseconds()*. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




**setUTCMinutes()**

Diese Methode ist funktional identisch mit `setMinutes()`. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




**setUTCMonth()**

Diese Methode ist funktional identisch mit `setMonth()`. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




**setUTCSeconds()**

Diese Methode ist funktional identisch mit `setSeconds()`. Der gesetzte Wert bedeutet jedoch nicht die lokale Zeit, sondern die Universal Coordinated Time (UTC), auch Greenwich Mean Time (GMT) genannt.




**setYear()**

Ändert das Jahr, das in Objektname gespeichert ist. Die Jahreszahl ist als Parameter zu übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zeit = new Date();
var AbsolutJetzt = Zeit.getTime();
var Jahr = Zeit.getYear();
Zeit.setYear(Jahr+1);
var AbsolutDann = Zeit.getTime();
var Differenz = AbsolutDann - AbsolutJetzt;
alert("bis zur gleichen Zeit in einem Jahr sind es " + Differenz + "ms");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel gibt die Anzahl der Millisekunden vom aktuellen Zeitpunkt bis zum gleichen Zeitpunkt in einem Jahr aus. Zu diesem Zweck wird ein neues Datumobjekt `Zeit` mit dem aktuellen Zeitpunkt erzeugt. Der Absolutzeitpunkt in Millisekunden wird mit `getTime()` ermittelt und in der Variablen `AbsolutJetzt` gespeichert. Mit `getYear()` wird das aktuelle Jahr ermittelt. Dann wird der Zeitpunkt mit `setYear()` um ein Jahr in die Zukunft verschoben. Der neue Zeitpunkt in Millisekunden wird wieder mit `getTime()` ermittelt. Das Ergebnis wird in der Variablen `AbsolutDann` gespeichert. Durch die Differenz zwischen den gespeicherten Werten von `AbsolutDann` und `AbsolutJetzt` wird die Anzahl der Millisekunden bis zum gleichen Zeitpunkt in einem Jahr ermittelt. Das Ergebnis wird in einem Meldungsfenster ausgegeben.




**toGMTString()**

Wandelt die in Objektname gespeicherten Daten in eine Zeichenkette nach dem IETF-Standard um. Wenn beispielsweise in Objektname der 2.10.1991 23:59:00 gespeichert ist, liefert diese Methode zurück: »Wed, 2 Oct 1991 23:59:00 GMT«, sofern sich der Benutzer in der entsprechenden Zeitzone befindet. Andernfalls wird die Zeitzonenverschiebung berücksichtigt. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var Jetzt = new Date();
  alert(Jetzt.toGMTString());
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt Jetzt mit dem aktuellen Zeitpunkt. Dann gibt es das Ergebnis der Methode Jetzt.toGMTString() als Meldungsfenster aus. Beispiel einer Ausgabe:

Wed, 11 Mar 1998 18:17:28 GMT (Netscape)

Wed, 11 Mar 1998 18:17:28 UTC (MS Internet Explorer)

**Beachten Sie:** Die Sommerzeit wird von dieser Methode nicht berücksichtigt.




**toLocaleString()**

Wandelt die in Objektname gespeicherten Daten in eine Zeichenkette um. Die zurückgegebene Zeichenkette berücksichtigt die lokale Uhrzeit. Wenn beispielsweise in Objektname der 2.10.1991 23:59:00 gespeichert ist, liefert diese Methode zurück: »10/2/91 23:59«. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var Jetzt = new Date();
  alert(Jetzt.toLocaleString());
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel erzeugt ein neues Datumobjekt Jetzt mit dem aktuellen Zeitpunkt. Dann gibt es das Ergebnis der Methode Jetzt.toLocaleString() als Meldungsfenster aus. Beispiel einer Ausgabe: 03/11/1998 19:21:22




**UTC()**

Liefert die Anzahl Millisekunden zurück, die zwischen dem 1.1.1970 und einem gewünschten Zeitpunkt verstrichen sind. Der Zeitpunkt ist in Form von Parametern zu übergeben. Alle Para-

meter sind als Zahlenwerte zu übergeben. Die Angabe der Datumswerte ist zwingend, die der Uhrzeitwerte optional. Die Parameter im Einzelnen:

- *Jahr* = Jahreszahl
- *Monat* = Monatszahl zwischen 0 und 11
- *Tag* = Tageszahl zwischen 1 und 31
- *Stunden* = (optional) Stunde der Uhrzeit zwischen 0 und 23
- *Minuten* = (optional) Minuten der Uhrzeit zwischen 0 und 59
- *Sekunden* = (optional) Sekunden der Uhrzeit zwischen 0 und 59

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  document.write(Date.UTC(1998,2,11,19,26,00));
//-->
</script>
</body></html>
```

Das Beispiel schreibt mit `document.write()` die Anzahl der Millisekunden in die Datei, die zwischen dem 1.1.1970, 0.00 Uhr, und dem 11.3.1998, 19.26 Uhr vergangen sind.

## 20.24 Function

### Eigenschaften

|                        |                                  |
|------------------------|----------------------------------|
| <code>arguments</code> | (Argumentnamen-Array)            |
| <code>arity</code>     | (Anzahl Argumente)               |
| <code>caller</code>    | (Namen der aufrufenden Funktion) |

### Function: Allgemeines zur Verwendung

Über das Objekt `Function` haben Sie Zugriff auf Eigenschaften einer JavaScript-Funktion. JavaScript-Funktionen werden dadurch also selbst Gegenstand von JavaScript-Anweisungen. Sinnvoll ist das Arbeiten mit dem `Function`-Objekt beispielsweise im Zusammenhang mit einer variablen Anzahl von Parametern in einer Funktion. Ein neues Objekt `Function` können Sie extra erzeugen. Ein Beispiel:

```
Farbe = new Function("NeueFarbe","document.bgColor=NeueFarbe;");
```

Beispielentsprechung in normaler Schreibweise:

```
function Farbe(NeueFarbe) {
  document.bgColor=NeueFarbe;
}
```

Beispielaufruf für beide Fälle:

```
<a href="javascript:Farbe('#FFCC99')">Neue Hintergrundfarbe</a>
```

Betrachten Sie in dem Beispiel zunächst den mittleren Teil (»Beispielentsprechung in normaler Schreibweise«). Das ist die normale Definition einer Funktion. Im Teil darüber wird die gleiche Funktion definiert, jedoch in Form eines Funktionsobjekts. Das Funktionsobjekt wird mit `new Function(...)` erzeugt. Der Rückgabewert wird in einer Variablen gespeichert, im Beispiel in der Variablen `Farbe`. Als Parameter erwartet die `Function`-Funktion eine beliebige Anzahl Parameter, die bis zum vorletzten als Parameter interpretiert werden, die die Funktion erwarten soll. Im Beispiel gibt es einen solchen Parameter, nämlich den Parameter `NeueFarbe`. Der letzte Parameter, der der `Function`-Funktion übergeben wird, ist eine Zeichenkette, die den gesamten Funktionsinhalt darstellt, also die JavaScript-Anweisungen innerhalb der Funktion.

Eine Variable, in der eine Funktion gespeichert ist, wie im Beispiel die Variable `Farbe`, können Sie genauso aufrufen wie eine Funktion. Der dritte Teil (»Beispielaufruf für beide Fälle«) gilt tatsächlich für beide Fälle, denn im einen Fall wird die Variable `Farbe` aufgerufen, in der eine ganze Funktion gespeichert ist, und im anderen Fall wird eine Funktion namens `Farbe()` aufgerufen.

Sie müssen jedoch kein neues Funktionsobjekt erzeugen, sondern können auch direkt Eigenschaften des Funktionsobjekts auf eine Funktion anwenden.

Beispiel für direktes Verwenden von Funktionseigenschaften:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
function HTMLListe(Typ) {
  document.write("<" + Typ + ">");
  for (var i=1; i<HTMLListe.arguments.length; i++)
    document.write("<li>" + HTMLListe.arguments[i]+ "</li>");
  document.write("</" + Typ + ">");
}
HTMLListe("U","eins","zwei","drei");
//-->
</script>
</body></html>
```

Das Beispiel schreibt dynamisch eine HTML-Liste, entweder als Aufzählungsliste oder als nummerierte Liste, in die Datei. Dies geschieht in einer Funktion namens `HTMLListe()`. Unterhalb der Funktion steht der Aufruf der Funktion. Dabei werden vier Parameter übergeben. Erwarten tut die Funktion `HTMLListe()` jedoch nur einen Parameter, nämlich den Parameter `Typ`. Dabei sollte man ein `U` oder ein `0` übergeben. Abhängig davon erzeugt die Funktion eine `<ul>`-Liste (Aufzählungsliste) oder eine `<ol>`-Liste (nummerierte Liste).

Innerhalb der Funktion wird jedoch mit `HTMLListe.arguments.length` die tatsächliche Anzahl der übergebenen Elemente abgefragt. Die Funktion nimmt an, dass alle Parameter ab dem zweiten Elemente der gewünschten Liste sind. Die Funktion schreibt mit `document.write()`-Befehlen die entsprechenden Listenpunkte und setzt dabei reihenweise die übergebenen Parameter ein, da diese Aktion innerhalb einer `for`-Schleife erfolgt.

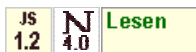
**arguments**

Speichert die Argumente, die einer Funktion übergeben wurden, in einem Array, also einer Kette von Elementen. Jedes Element stellt ein Argument dar. Die Eigenschaft ist nur innerhalb der Funktion verfügbar, auf die sie sich bezieht. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script language="JavaScript" type="text/javascript">
<!--
function Test(Name,Vorname) {
  document.write("Funktion bekam " + Test.arguments.length + " Argumente");
  for (var i=0; i<Test.arguments.length; ++i)
    document.write("<BR>" + Test.arguments[i]);
}
Test("Muenz", "Stefan");
//-->
</script>
</body></html>
```

Das Beispiel definiert eine Funktion `Test()`, die zwei Parameter übergeben bekommt. Die Funktion schreibt dynamisch in die HTML-Datei, wie viele Parameter sie erwartet, und danach in einer `for`-Schleife, welche Werte übergeben wurden.

Mit `Funktionsname.arguments.length` ermitteln Sie die Anzahl der tatsächlich übergebenen Parameter (nicht die Anzahl der erwarteten Parameter). Mit `Funktionsname.arguments[0]` sprechen Sie den Wert des ersten übergebenen Parameters an, mit `Funktionsname.arguments[1]` den des zweiten Parameters usw.

**arity**

Speichert die Anzahl Argumente, die eine Funktion erwartet. Die Eigenschaft ist nur außerhalb der Funktion verfügbar, auf die sie sich bezieht. Der Script-Bereich, in dem die Anweisung vorkommt, muss mit `JavaScript1.2` ausgezeichnet sein. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript1.2" type="text/javascript">
<!--
function Test(Name,Vorname) {
  return(Vorname + " " + Name);
}
alert("Funktion erwartet " + Test.arity + " Argumente");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert eine Funktion `Test()`, die zwei Parameter erwartet. Die Funktion wird im Beispiel gar nicht aufgerufen, dafür wird in einem Meldungsfenster ausgegeben, wie viele Argumente die Funktion `Test()` erwartet. Dazu wird die Eigenschaft `arity` auf die gewünschte Funktion angewendet.





Speichert die gesamte Funktion, von der aus die aktuelle Funktion aufgerufen wird. Hat den Wert `null`, falls der Aufruf direkt von einem Event-Handler oder beim Einlesen der Datei von außerhalb einer anderen Funktion erfolgte. Die Eigenschaft ist nur innerhalb der Funktion verfügbar, auf die sie sich bezieht. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Test() {
    alert(Test.caller);
}
function Aufruf() {
    Test();
}
//-->
</script>
</head><body>
<a href="javascript:Aufruf()">Wer ruft Test()?</a>
</body></html>
```

Das Beispiel definiert im Dateikopf zwei Funktionen, eine Funktion `Test()` und eine Funktion `Aufruf()`. Die Funktion `Test()` tut nichts anderes als auszugeben, von wo sie aufgerufen wurde, die Funktion `Aufruf()` nichts anderes, als `Test()` aufzurufen. In der Datei ist ferner ein Verweis notiert. Beim Anklicken des Verweises wird die Funktion `Aufruf()` aufgerufen, die wiederum `Test()` aufruft, woraufhin `Test()` den gesamten Code der Funktion `Aufruf()` ausgibt.

**Beachten Sie:** Opera 5.12 interpretiert diese Eigenschaft nicht.

## 20.25 Math

### Eigenschaften

|         |                                       |
|---------|---------------------------------------|
| E       | (Eulersche Konstante)                 |
| LN2     | (natürlicher Logarithmus von 2)       |
| LN10    | (natürlicher Logarithmus von 10)      |
| LOG2E   | (konstanter Logarithmus von 2)        |
| LOG10E  | (konstanter Logarithmus von 10)       |
| PI      | (Konstante PI)                        |
| SQRT1_2 | (Konstante für Quadratwurzel aus 0,5) |
| SQRT2   | (Konstante für Quadratwurzel aus 2)   |

### Methoden

|                     |                  |
|---------------------|------------------|
| <code>abs()</code>  | (positiver Wert) |
| <code>acos()</code> | (Arcus Cosinus)  |
| <code>asin()</code> | (Arcus Sinus)    |

|                       |                                         |
|-----------------------|-----------------------------------------|
| <code>atan()</code>   | (Arcus Tangens)                         |
| <code>ceil()</code>   | (nächsthöhere ganze Zahl)               |
| <code>cos()</code>    | (Cosinus)                               |
| <code>exp()</code>    | (Exponentialwert)                       |
| <code>floor()</code>  | (nächstniedrigere ganze Zahl)           |
| <code>log()</code>    | (Anwendung des natürlichen Logarithmus) |
| <code>max()</code>    | (größere von zwei Zahlen)               |
| <code>min()</code>    | (kleinere von zwei Zahlen)              |
| <code>pow()</code>    | (Zahl hoch Exponent)                    |
| <code>random()</code> | (0 oder 1 per Zufall)                   |
| <code>round()</code>  | (kaufmännische Rundung einer Zahl)      |
| <code>sin()</code>    | (Sinus)                                 |
| <code>sqrt()</code>   | (Quadratwurzel)                         |
| <code>tan()</code>    | (Tangens)                               |

### Math: Allgemeines zur Verwendung

Mit dem Objekt `Math` können Sie Berechnungen, auch komplexe wissenschaftlicher oder kaufmännischer Art, durchführen. Dazu stehen Ihnen verschiedene mächtige Methoden und Funktionen sowie einige Eigenschaften zur Verfügung. Eine Instanz von `Math` brauchen Sie nicht eigens erzeugen. Sie können Eigenschaften und Methoden von `Math` direkt verwenden.

Schema:

```
x = Math.Eigenschaft;
x = Math.Methode(Parameter);
```

Mit `Zahl = 10 * Math.PI` beispielsweise steht in der Variablen `Zahl` hinterher das Produkt aus der Zahl `PI` und `10`. Mit `Wurzel = Math.sqrt(10)` steht in der Variablen `Wurzel` hinterher das Ergebnis der Quadratwurzel aus `10`. Notieren Sie vor jedem Aufruf einer Eigenschaft oder Methode des `Math`-Objekts `Math` (großgeschrieben).

Bei jedem Zahlenparameter, den Sie einer Methode von `Math` übergeben, kann es sich um eine feste Zahl (z.B. `25` oder `0.123`) handeln, um eine numerische Variable (z.B. `x`) oder um einen Rechenausdruck (z.B. `7 * 5 + 0.3`). Auch Rechenausdrücke mit Variablen sind erlaubt (z.B. `x * i + 1`).

Beachten Sie, dass die Anzahl der Nachkommastellen in Programmiersprachen wie JavaScript systembedingt beschränkt ist (auf den heutigen Systemen haben solche Zahlen normalerweise 32 Bit Länge). JavaScript unterscheidet nicht wie manche andere Programmiersprachen zwischen verschiedenen Typen numerischer Werte (Integer, Float usw.). Eventuell erforderliche Typumwandlungen erfolgen automatisch.



Die Eulersche Konstante – Wert ca. 2,718. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
```

```
<!--
document.write(Math.E);
//-->
</script>
</body></html>
```

JS 1.0 N 2.0 3.0 Lesen LN2

Der natürliche Logarithmus von 2 – Konstante mit einem Wert von ca. 0,693. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write(Math.LN2);
//-->
</script>
</body></html>
```

JS 1.0 N 2.0 3.0 Lesen LN10

Der natürliche Logarithmus von 10 – Konstante mit einem Wert von ca. 2,302. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write(Math.LN10);
//-->
</script>
</body></html>
```

JS 1.0 N 2.0 3.0 Lesen LOG2E

Logarithmus von 2 – Konstante mit einem Wert von ca. 1,442. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write(Math.LOG2E);
//-->
</script>
</body></html>
```





**LOG10E**

Logarithmus von 10 – Konstante mit einem Wert von ca. 0,434. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write(Math.LOG10E);
//-->
</script>
</body></html>
```





**PI**

Konstante Pi für Kreisberechnungen mit einem Wert von ca. 3,14159. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write(Math.PI);
//-->
</script>
</body></html>
```





**SQRT1\_2**

Konstante für Quadratwurzel aus 0,5 – Wert von ca. 0,707. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write(Math.SQRT1_2);
//-->
</script>
</body></html>
```





**SQRT2**

Konstante für Quadratwurzel aus 2 – Wert von ca. 1,414. Das Beispiel schreibt dynamisch den Wert der Eigenschaft an die aktuelle Stelle der HTML-Datei.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
```

```
document.write(Math.SQRT2);
//-->
</script>
</body></html>
```

### **abs()**

Erwartet als Parameter eine Zahl. Liefert in jedem Fall den positiven Wert der Zahl zurück. War dieser vorher negativ, wird er positiv. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="=" onClick="Test.Aus.value=Math.abs(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `abs()` auf den Wert aus dem ersten Eingabefeld. Wird im ersten Feld eine negative Zahl eingegeben, erscheint im zweiten Feld die gleiche Zahl, allerdings positiv.

### **acos()**

Erwartet als Parameter eine Zahl. Liefert den Arcus Cosinus dieser Zahl zurück. Achten Sie darauf, dass Sie sinnvolle Werte zwischen -1 und +1 übergeben. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="=" onClick="Test.Aus.value=Math.acos(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `acos()` auf den Wert aus dem ersten Eingabefeld.

**Beachten Sie:** Diese Methode erwartet Zahlen mit der Einheit *Radian* (rad) als Parameter.

### **asin()**

Erwartet als Parameter eine Zahl. Liefert den Arcus Sinus dieser Zahl zurück. Achten Sie darauf, dass Sie sinnvolle Werte zwischen -1 und +1 übergeben. Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="=" onClick="Test.Aus.value=Math.asin(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `asin()` auf den Wert aus dem ersten Eingabefeld.

**Beachten Sie:** Diese Methode erwartet Zahlen in der Einheit *Radian* (rad) als Parameter.



Erwartet als Parameter eine Zahl. Liefert den Arcus Tangens dieser Zahl zurück. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.atan(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `atan()` auf den Wert aus dem ersten Eingabefeld.

**Beachten Sie:** Diese Methode erwartet Zahlen in der Einheit *Radian* (rad) als Parameter.



Erwartet als Parameter eine Zahl. Liefert die nächsthöhere Ganzzahl dieser Zahl zurück (Aufrundung). Wenn die übergebene Zahl eine Ganzzahl ist, bleibt sie unverändert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.ceil(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `ceil()` auf den Wert aus dem ersten Eingabefeld. Bei Eingabe von Bruchzahlen im ersten Eingabefeld steht im zweiten Feld anschließend die nächsthöhere Ganzzahl.



Erwartet als Parameter eine Zahl. Liefert den Cosinus dieser Zahl zurück. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.cos(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `cos()` auf den Wert aus dem ersten Eingabefeld.

**Beachten Sie:** Diese Methode erwartet Zahlen in der Einheit *Radian* (rad) als Parameter.



Erwartet als Parameter eine Zahl. Liefert den Exponentialwert dieser Zahl auf Basis der Eulerschen Konstanten zurück. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.exp(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `exp()` auf den Wert aus dem ersten Eingabefeld.

**Beachten Sie:** Sobald Sie eine Zahl verwenden, die größer als 709 ist, liefert diese Methode *infinity* (Zahl ist zu groß) zurück. Ist die Zahl kleiner als -744, liefert die Methode 0 zurück.



Erwartet als Parameter eine Zahl. Liefert die nächstniedrigere Ganzzahl dieser Zahl zurück (Abrundung). Wenn die übergebene Zahl eine Ganzzahl ist, bleibt sie unverändert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.floor(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `floor()` auf den Wert aus dem ersten Eingabefeld. Bei Eingabe von Bruchzahlen im ersten Eingabefeld steht im zweiten Feld anschließend die nächstniedrigere Ganzzahl. Bei positiven Zahlen ist dies die Zahl ohne Nachkommastellen.



Erwartet als Parameter eine Zahl. Liefert die Anwendung des natürlichen Logarithmus auf diese Zahl zurück. Achten Sie darauf, dass Sie sinnvolle Werte größer als 0 übergeben.

Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.log(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `log()` auf den Wert aus dem ersten Eingabefeld.



Erwartet zwei Zahlen als Parameter. Gibt die größere der beiden Zahlen zurück. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input name="z1"><input name="z2"><input name="Aus">
<input type="button" value=""
onClick="Test.Aus.value=Math.max(Test.z1.value,Test.z2.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit drei Eingabefeldern und einem Button. Nach Eingabe je einer Zahl im ersten und im zweiten Eingabefeld und Klick auf den Button wird im dritten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `max()` auf die Werte aus den beiden ersten Eingabefeldern.



Erwartet zwei Zahlen als Parameter. Gibt die kleinere der beiden Zahlen zurück. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input name="z1"><input name="z2"><input name="Aus">
<input type="button" value=""
onClick="Test.Aus.value=Math.min(Test.z1.value,Test.z2.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit drei Eingabefeldern und einem Button. Nach Eingabe je einer Zahl im ersten und im zweiten Eingabefeld und Klick auf den Button wird im dritten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `min()` auf die Werte aus den beiden ersten Eingabefeldern.






**pow()**

Erwartet zwei Zahlen als Parameter. Die erste Zahl wird als Basis interpretiert, die zweite als Exponent. Liefert den Wert von »Zahl« hoch »Exponent« zurück. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action="">
<input name="z1"><input name="z2"><input name="Aus">
<input type="button" value=""
onClick="Test.Aus.value=Math.pow(Test.z1.value,Test.z2.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit drei Eingabefeldern und einem Button. Nach Eingabe je einer Zahl im ersten und im zweiten Eingabefeld und Klick auf den Button wird im dritten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `pow()` auf die Werte aus den beiden ersten Eingabefeldern. Wird im ersten Eingabefeld beispielsweise 2 (Basis) und im zweiten Eingabefeld 3 (Exponent) eingegeben, erscheint im dritten Feld als Ausgabe 8.




**random()**

Gibt eine Zufallszahl zwischen 0 und 1 zurück. Es handelt sich um eine Bruchzahl, z.B. 0.3834417857195922. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form action="">
<input type="button" value="Button" onClick="alert(Math.random())">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit einem Button. Beim Klicken auf den Button wird eine Zufallszahl zwischen 0 und 1 ausgegeben. Es handelt sich dabei um eine Pseudo-Zufallszahl, wie sie auf heute üblichen EDV-Systemen realisierbar ist. Erst seit Netscape 3.x auf allen Plattformen verfügbar. Bei Netscape 2.x nur unter Unix.




**round()**

Erwartet als Parameter eine Zahl. Liefert die kaufmännisch gerundete Ganzzahl dieser Zahl zurück. Wenn die übergebene Zahl eine Ganzzahl ist, bleibt sie unverändert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="" onClick="Test.Aus.value=Math.round(Test.Ein.value)">
</form>
</body></html>
```

Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `round()` auf den Wert aus dem ers-

ten Eingabefeld. Bei Eingabe von Bruchzahlen im ersten Eingabefeld steht im zweiten Feld anschließend die kaufmännisch gerundete Ganzzahl.



Erwartet als Parameter eine Zahl. Liefert den Sinus dieser Zahl zurück. Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `sin()` auf den Wert aus dem ersten Eingabefeld:

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="=" onClick="Test.Aus.value=Math.sin(Test.Ein.value)">
</form>
</body></html>
```

**Beachten Sie:** Diese Methode erwartet Zahlen in der Einheit *Radiant* (rad) als Parameter.



Erwartet als Parameter eine Zahl. Liefert die Quadratwurzel dieser Zahl zurück. Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `sqrt()` auf den Wert aus dem ersten Eingabefeld.

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="=" onClick="Test.Aus.value=Math.sqrt(Test.Ein.value)">
</form>
</body></html>
```



Erwartet als Parameter eine Zahl. Liefert den Tangens dieser Zahl zurück. Das Beispiel definiert ein Formular mit zwei Eingabefeldern und einem Button. Nach Eingabe einer Zahl im ersten Eingabefeld und Klick auf den Button wird im zweiten Eingabefeld das Ergebnis ausgegeben. Das Ergebnis ist die Anwendung von `tan()` auf den Wert aus dem ersten Eingabefeld.

```
<html><head><title>Test</title>
</head><body>
<form name="Test" action=""><input name="Ein"><input name="Aus">
<input type="button" value="=" onClick="Test.Aus.value=Math.tan(Test.Ein.value)">
</form>
</body></html>
```

**Beachten Sie:** Diese Methode erwartet Zahlen in der Einheit *Radiant* (rad) als Parameter.

## 20.26 navigator

### Eigenschaften

|               |                                        |
|---------------|----------------------------------------|
| appName       | (Spitzname des Browsers)               |
| appName       | (offizieller Name des Browsers)        |
| appVersion    | (Browser-Version)                      |
| cookieEnabled | (Cookies erlaubt)                      |
| language      | (Browser-Sprache)                      |
| platform      | (Plattform, auf der der Browser läuft) |
| userAgent     | (HTTP-Identifikation des Browsers)     |

### Methoden

|               |                                 |
|---------------|---------------------------------|
| javaEnabled() | (Java-Verfügbarkeit überprüfen) |
|---------------|---------------------------------|

### Unterobjekte

mimeTypes  
plugins

### navigator: Allgemeines zur Verwendung

Über das Objekt `navigator` können Sie in einem JavaScript Informationen darüber ermitteln, welchen Web-Browser der Anwender verwendet, sowie einige nähere Spezifikationen dazu. Das kann zum Beispiel interessant sein, um die Ausführung von JavaScript-Anweisungen davon abhängig zu machen, welchen Browser der Anwender benutzt. So lassen sich Fehlermeldungen bei Anwendern vermeiden, die einen Browser verwenden, der bestimmte JavaScript-Befehle nicht kennt. Eigenschaften und Methoden von `navigator` können Sie direkt ansprechen. Beispiel: `navigator.appName`.



#### appName

Speichert den Spitznamen des Browsers. Dieses Feature wurde von Netscape deshalb erfunden, weil der Netscape-Browser den Spitznamen »Mozilla« trägt. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(navigator.appCodeName == "Mozilla")
  alert("So ein drachenstarker Browser!");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel ermittelt beim Einlesen der Datei, ob der vom Anwender benutzte Browser auf den Spitznamen »Mozilla« hört. Wenn ja, wird die Meldung »So ein affenartiger Browser!« ausgegeben. Sowohl Netscape als auch der MS Internet Explorer wie auch Opera 5.12 speichern in dieser Eigenschaft den Wert »Mozilla«.

**appName**

Speichert den Namen des Browsers, den der Anwender verwendet. Netscape speichert in dieser Eigenschaft beispielsweise den Wert »Netscape«, der MS Internet Explorer den Wert »Microsoft Internet Explorer«. Das Beispiel schreibt mit der `write()`-Methode in die HTML-Datei, welchen Browser der Anwender verwendet:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.write("So, so, Sie verwenden also " + navigator.appName);
//-->
</script>
</body></html>
```

**Beachten Sie:** Unter Opera kann der Anwender frei wählen, wie sich der Browser identifiziert.

**appVersion**

Speichert die Version des Browsers, den der Anwender verwendet. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(navigator.appVersion.substring(0,1) == "4")
  alert("Oh, ein Browser der 4. Generation!");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel fragt ab, ob das erste Zeichen aus dem Wert, der in `navigator.appVersion` gespeichert ist, eine 4 ist. Wenn ja, wird eine Meldung ausgegeben. Der Wert von `navigator.appVersion` umfasst mehr als nur die reine Produktversion. Netscape 4.01 liefert z.B. folgenden Wert: "4.01 [en] (Win95; I)", der MS Internet Explorer 4.0 liefert z.B.: "4.0 (compatible; MSIE 4.0; Windows 95)". Um Teile aus einer solchen Zeichenkette zu extrahieren wie im Beispiel mit der Methode `substring()`, lesen Sie den Abschnitt über das `string`-Objekt.

**Beachten Sie:** Unter Opera kann der Anwender frei wählen, wie sich der Browser identifiziert. Damit können je nach Nutzereinstellung auch in dieser Eigenschaft beliebige und eventuell unrealistische Werte gespeichert sein.

**cookieEnabled**

Speichert, ob der Nutzer das Setzen von Cookies erlaubt. Sind diese erlaubt, so enthält die Eigenschaft `true`, sonst `false`. Nicht geprüft werden kann mit dieser Eigenschaft, ob der Nutzer vor dem Speichern eine Warnmeldung erhält. In diesem Fall ist in der Eigenschaft ebenfalls `true` gespeichert. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
```

```

<!--
  if(navigator.cookieEnabled == true)
    document.write("Cookies erlaubt");
  else if(navigator.cookieEnabled == false) document.write("Cookies verboten.");
    else document.write("Verrate ich nicht.");

//-->
</script>
</body></html>

```

Im Beispiel wird gefragt, ob der Nutzer Cookies erlaubt. Ist das der Fall, wird mit der `write()`-Methode eine entsprechende Information ausgegeben. Sind keine Cookies verfügbar, wird gefragt, ob Cookies verboten sind. Browser, die `cookieEnabled` kennen und keine Cookies erlauben, folgen dem `if`-Zweig der Anweisung. Browser, die diese Eigenschaft nicht kennen, geben den Text "Verrate ich nicht." aus.

### JS 1.2 N 4.0 Lesen *language*

Speichert die Sprache der Benutzerführung des Browsers, den der Anwender verwendet. Die Sprache wird dabei mit den international üblichen Abkürzungen gespeichert, z.B. `en` für englisch oder `de` für deutsch. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  if(navigator.language.indexOf("en")>-1)
    document.write("dear vistor, welcome on our pages");
  if(navigator.language.indexOf("de")>-1)
    document.write("Sehr geehrter Besucher, willkommen auf unseren Seiten");
//-->
</script>
</body></html>

```

Das Beispiel ermittelt, welche Benutzersprache der Browser des Anwenders verwendet. Wenn es englisch ist, wird mit der `write()`-Methode ein englischer Begrüßungstext in die HTML-Datei geschrieben, wenn es deutsch ist, ein deutscher Begrüßungstext.

**Beachten Sie:** Im MS Internet Explorer heißt diese Eigenschaft `navigator.userLanguage`.

### JS 1.2 N 4.0 Lesen *platform*

Speichert, auf welchem Betriebssystemtyp der Browser läuft, den der Anwender verwendet. Das Beispiel gibt beim Einlesen der Datei in einem Meldungsfenster aus, auf welcher Plattform der Browser beim Anwender läuft. Bei Windows 95 gibt Netscape beispielsweise »Win32« aus.

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert(navigator.platform);
//-->

```

```

</script>
</head><body>
</body></html>

```

**userAgent**

Speichert alle wichtigen Informationen über den Browser, den der Anwender verwendet, und zwar in der Form, die der Browser auch bei jeder HTTP-Anforderung einer Datei im WWW sendet. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
Diese Daten sendet Ihr Browser, wenn er Dateien im WWW anfordert:<br>
<script type="text/javascript">
<!--
  document.write(navigator.userAgent);
//-->
</script>
</body></html>

```

Das Beispiel schreibt in die HTML-Datei, mit welchen Daten sich der verwendete Browser im WWW ausgibt. Im Anhang A.4 *Client-Identifikationen* finden Sie eine Übersicht bekannter Ausgaben, die in dieser Eigenschaft enthalten sein können. Die Übersicht kann helfen, wenn Sie Abfragen nach bestimmten Browsern, Versionen oder Plattformen durchführen wollen.

**Beachten Sie:** Unter Opera kann der Anwender frei wählen, wie sich der Browser identifiziert. Damit sind je nach Nutzereinstellung in dieser Eigenschaft verschiedene Werte gespeichert. Unabhängig von den Nutzereinstellungen ist jedoch in dieser Eigenschaft immer das Wort *Opera* enthalten, wenn auch an verschiedenen Positionen.

**javaEnabled()**

Testet, ob Java auf dem Browser des Anwenders verfügbar ist oder nicht. Liefert *true* zurück, wenn Java verfügbar ist, und *false*, wenn nicht. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  if(navigator.javaEnabled())
  {
    document.write("<applet code=\"zticker.class\" width=\"150\" height=\"25\">");
    document.write("</applet>");
  }
//-->
</script>
</body></html>

```

Das Beispiel fragt mit `if(navigator.javaEnabled())` ab, ob Java verfügbar ist. Wenn ja, wird mit der `write()`-Methode der HTML-Code für die Platzierung eines Java-Applets geschrieben.

## 20.27 *mimeTypes*

### **Eigenschaften**

|                             |                                 |
|-----------------------------|---------------------------------|
| <code>description</code>    | (Beschreibung eines Mime-Types) |
| <code>enabledPlug-In</code> | (Plug-In vorhanden)             |
| <code>length</code>         | (Anzahl Mime-Typen)             |
| <code>suffixes</code>       | (Dateiendungen)                 |
| <code>type</code>           | (Mime-Type)                     |

### ***mimeTypes: Allgemeines zur Verwendung***

Über das Objekt `mimeTypes`, das in der JavaScript-Objekthierarchie unterhalb des `navigator`-Objekts liegt, können Sie ermitteln, welche Dateitypen der Browser des Anwenders kennt, und ob ein Plug-In zum Anzeigen oder Abspielen eines Dateityps vorhanden ist. Lesen Sie zum besseren Verständnis von Mime-Typen gegebenenfalls den Anhang A.2. Um einen Mime-Type anzusprechen, stehen folgende Möglichkeiten zur Verfügung:

Schema 1:

```
navigator.mimeTypes[#].Eigenschaft
```

Schema 2:

```
navigator.mimeTypes["MimeType"].Eigenschaft
```

Mime-Typen können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1): Bei Verwendung von Indexnummern geben Sie `navigator.mimeTypes` an und dahinter in eckigen Klammern eine Zahl. Sie können allerdings nicht wissen, welche Mime-Typen an welcher Stelle gespeichert sind. Die Verwendung von Indexnummern ist deshalb nur sinnvoll, wenn Sie beispielsweise in einer Schleife alle verfügbaren Mime-Typen ermitteln.
- Mit dem Namen eines Mime-Typs als Indexnamen (wie in Schema 2): Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `navigator.mimeTypes` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Mime-Type-Namen. Dabei gilt die übliche Notation für Mime-Typen, also die Form »Kategorie/Unterkategorie«. Als Indexnamen sind zum Beispiel Werte möglich, die Sie mit der Objekteigenschaft `type` ermitteln können.

Beispiel für das Objekt insgesamt:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.writeln("<table border='1'>");
for(var i=0; i<navigator.mimeTypes.length; i++)
{
  document.writeln("<tr>");
  document.writeln("<td>" + navigator.mimeTypes[i].type + "<\td>");
  document.writeln("<td>" + navigator.mimeTypes[i].suffixes + "<\td>");
```

```

document.writeln("<td>" + navigator.mimeTypes[i].description + "<\td>");
if (navigator.mimeTypes[i].enabledPlug-In)
    document.writeln("<td>" + navigator.mimeTypes[i].enabledPlug-In + "<\td>");
else document.writeln("<td>kein Plug-In<\td>");
document.writeln("<\tr>");
}
document.writeln("<\table>");
//-->
</script>
</body></html>

```

Das Beispiel schreibt mit Hilfe der Methode `writeln()` dynamisch eine Tabelle in die HTML-Datei. In der Tabelle werden alle verfügbaren Plug-Ins aufgelistet. In der linken Spalte steht der Mime-Type Typ, also die Kategorie/Unterkategorie, wie Sie sie auch zur Mime-Type-Adressierung mit Hilfe von Indexnamen verwenden können. In der Spalte daneben stehen die zugehörigen Dateiendungen. Daneben steht eine Kurzbeschreibung des Mime-Type, und in der letzten Spalte steht, ob der Browser ein Plug-In besitzt, um Dateien des entsprechenden Typs zu verarbeiten.

**Beachten Sie:** Welche Eigenschaften im Objekt `mimeTypes` gespeichert sind, ist stark abhängig vom verwendeten Browser. So speichert der Netscape 6 nur tatsächlich vorhandene Plug-Ins als eigene Objekteigenschaft, während Opera 5.12 und Netscape 4 z.B. auch den Type `image/jpeg` mit angeben. Im MS Internet Explorer existiert dieses Objekt nur aus Kompatibilitätsgründen und enthält keinen Inhalt.

#### Lesen *description*

Speichert die Kurzbeschreibung eines Mime-Typs. Das Beispiel prüft zuerst, ob der Mime-Type `application/pdf` existiert. Wenn ja, wird in einem Meldungsfenster die Kurzbeschreibung zum Mime-Type `application/pdf` ausgegeben. Netscape 4.x gibt dabei freilich nicht viel mehr bekannt als »Acrobat«, in Opera 5.12 und Netscape 6 wird `Acrobat (*.pdf)` ausgegeben.

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
if (navigator.mimeTypes["application/pdf"])
    alert(navigator.mimeTypes["application/pdf"].description);
//-->
</script>
</head><body>
</body></html>

```

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.

#### Lesen *enabledPlug-In*

Speichert, ob für einen Mime-Type ein Plug-In installiert ist. Wenn kein Plug-In verfügbar ist, der Browser jedoch den Mime-Type kennt, wird der Wert `null` gespeichert. Die Eigenschaft stellt selbst ein Objekt dar. Sie ist dabei eine Art Zeiger auf das entsprechende `plugins`-Objekt. Hinter der Eigenschaft lassen sich wiederum alle Eigenschaften und Methoden des `plugins`-Objekts notieren. Ein Beispiel:



```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
if (navigator.mimeTypes["x-world/x-vrml"])
  if(navigator.mimeTypes["x-world/x-vrml"].enabledPlugin != null)
    document.write("<object data=\"yzeplin.wrl\" width=\"400\"
height=\"300\"></object>");
//-->
</script>
</body></html>

```

Das Beispiel ermittelt, ob der Browser den Mime-Type kennt und ob er ein Plug-In für VRML-Dateien verfügbar hat. Wenn ja, wird mit Hilfe der `write()`-Methode dynamisch das HTML-Konstrukt zum Einbinden einer VRML-Datei geschrieben. Anwender, die kein VRML-Plug-In haben, bekommen dadurch keine Fehlermeldungen, sondern ein leeres Objektfenster. Solche Anwender bekommen einfach gar nicht mit, dass in der Datei eigentlich eine VRML-Datei referenziert ist.

#### Lesen *length*

Speichert die Anzahl der Mime-Type-Einträge, die der Browser kennt. Das Beispiel fragt in einer `for`-Schleife ab, ob für den jeweils aktuellen Mime-Type ein Plug-In installiert ist. Wenn ja, wird eine Zeile mit der Kurzbeschreibung des Mime-Types geschrieben. So werden am Ende alle installierten Plug-Ins mit sprechenden Beschreibungen aufgelistet. Bei der Definition der `for`-Schleife dient die Eigenschaft `navigator.mimeTypes.length` als Abbruchbedingung.

```

<html><head><title>Test</title>
</head><body>
<b>Folgende Plug-Ins sind installiert:</b>
<script type="text/javascript">
<!--
for(var i=0; i<navigator.mimeTypes.length; ++i)
  if(navigator.mimeTypes[i].enabledPlugin != null)
    document.write("<br>" + navigator.mimeTypes[i].description);
//-->
</script>
</body></html>

```

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.

#### Lesen *suffixes*

Speichert die typischen Dateiendungen, die zu einem Mime-Type gehören. Das Beispiel prüft, ob der Browser den Mime-Type kennt, und gibt in einem Meldungsfenster aus, welche Dateiendungen unter den Mime-Type `image/jpeg` fallen.

```

<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
if (navigator.mimeTypes["image/jpeg"])
  alert(navigator.mimeTypes["image/jpeg"].suffixes);

```

```
//-->
</script>
</head><body>
</body></html>
```

**Beachten Sie:** Im Netscape 6 ist dieses Beispiel nicht nachvollziehbar, da dieser nur die Mime-Typen als eigene Objekteigenschaft speichert, zu denen tatsächlich ein Plug-In vorliegt. Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.

**JS** **N** **Lesen** **type**  
1.1 3.0

Speichert die eigentliche Mime-Type-Bezeichnung nach dem Schema »Kategorie/Unterkategorie«. Das Beispiel klappert in einer `for`-Schleife alle dem Browser bekannten Mime-Typen ab und schreibt jeweils eine Zeile mit der Mime-Type-Bezeichnung. So werden am Ende alle dem Browser bekannten Mime-Types aufgelistet.

```
<html><head><title>Test</title>
</head><body>
<b>Folgende Mime-Typen sind dem Browser bekannt:</b>
<script type="text/javascript">
<!--
for(var i=0; i<navigator.mimeTypes.length; ++i)
  document.write("<br>" + navigator.mimeTypes[i].type);
//-->
</script>
</body></html>
```

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.

## 20.28 plugins

### Eigenschaften

|                          |                              |
|--------------------------|------------------------------|
| <code>description</code> | (Beschreibung eines Plug-In) |
| <code>filename</code>    | (Dateiname eines Plug-In)    |
| <code>length</code>      | (Anzahl Plug-In)             |
| <code>name</code>        | (Produktname eines Plug-In)  |

### plugins: Allgemeines zur Verwendung

Über das Objekt `plugins`, das in der JavaScript-Objekthierarchie unterhalb des `navigator`-Objekts liegt, können Sie ermitteln, welche Plug-Ins im Sinne der Netscape-Plug-In-Technik des Anwenders installiert hat. Um ein Plug-In anzusprechen, stehen folgende Möglichkeiten zur Verfügung:

Schema 1:

```
navigator.plugins[#].Eigenschaft
```

Schema 2:

```
navigator.plugins["Name"].Eigenschaft
```

Plug-Ins können Sie auf zwei Arten ansprechen:

- Mit einer Indexnummer (wie in Schema 1): Bei Verwendung von Indexnummern geben Sie `navigator.plugins` an und dahinter in eckigen Klammern eine Zahl. Sie können allerdings nicht wissen, welche Plug-Ins an welcher Stelle gespeichert sind. Die Verwendung in Verbindung mit Indexnummern ist deshalb nur sinnvoll, wenn Sie beispielsweise in einer Schleife alle verfügbaren Plug-Ins ermitteln.
- Mit dem Namen eines Plug-In als Indexnamen (wie in Schema 2): Dabei notieren Sie wie beim Ansprechen mit Indexnummer hinter `navigator.plugins` eckige Klammern. Innerhalb der eckigen Klammern notieren Sie in Anführungszeichen den Namen des Plug-In. Als Namen sind zum Beispiel Werte möglich, die Sie mit der Objekteigenschaft `name` ermitteln können.

Beispiel für das Objekt insgesamt:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
document.writeln("<table border='1'">");
for(var i=0; i<navigator.plugins.length; i++)
{
  document.writeln("<tr>");
  document.writeln("<td>" + navigator.plugins[i].name + "</td>");
  document.writeln("<td>" + navigator.plugins[i].description + "</td>");
  document.writeln("<td>" + navigator.plugins[i].filename + "</td>");
  document.writeln("</tr>");
}
document.writeln("</table>");
//-->
</script>
</body></html>
```

Das Beispiel schreibt mit Hilfe der Methode `writeln()` dynamisch eine Tabelle in die HTML-Datei. In der Tabelle werden alle verfügbaren Plug-Ins aufgelistet. In der linken Spalte steht der Name des Plug-In, wie Sie ihn auch bei Plug-In-Adressierungen mit Indexnamen benutzen können. In der Spalte daneben steht eine Kurzbeschreibung, und in der letzten Spalte steht, welche Datei auf dem Rechner des Anwenders das Plug-In repräsentiert.

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.

**JS** **N** **Lesen** *description*  
1.1 3.0

Speichert die Kurzbeschreibung eines Plug-In. Das Beispiel fragt ab, ob ein Plug-In mit dem Namen »Java Plug-in« beim Anwender installiert ist (das Beispiel nimmt an, dass der Autor dieses Plug-In besitzt und testen möchte, ob er seine Daten zu diesem Plug-In einem Anwender zumuten kann). Das Beispiel beschränkt sich darauf, in einem Meldungsfenster die Kurzbeschreibung des Plug-In auszugeben, wenn dieses Plug-In installiert ist.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  if(navigator.plugins["Java Plug-in"])
    alert(navigator.plugins['Java Plug-in'].description);
//-->
</script>
</head><body>
</body></html>
```

JS  
1.1N  
3.0

Lesen

**filename**

Speichert den Namen der Programmdatei eines Plug-In. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(navigator.plugins["VDOLive small plugin"].filename.indexOf("npsmlvdo.dll"))
  alert("Plug-In ist eine Nummer zu klein");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel nimmt an, dass zuvor ermittelt wurde, dass ein Plug-In mit dem Namen "VDOLive small plugin" beim Anwender installiert ist. Nun fragt das Beispiel ab, ob in dem Eintrag, der den Dateinamen des Plug-In speichert, der Wert "npsmlvdo.dll" vorkommt. Wenn ja, wird im Beispiel die Meldung ausgegeben, dass das Plug-In eine Nummer zu klein ist. In der Eigenschaft `filename` speichert Netscape den vollständigen Pfadnamen der Datei. Um den eigentlichen Dateinamen abzufragen, wird im Beispiel eine Methode des `string`-Objekts bemüht, nämlich `indexOf()`.

JS  
1.1N  
3.0

Lesen

**length**

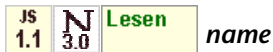
Speichert die Anzahl der Plug-Ins, die der Anwender zu seinem Browser installiert hat. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<b>Folgende Plug-In-Dateien sind auf dem Rechner:</b>
<script type="text/javascript">
<!--
for(var i=0; i<navigator.plugins.length; ++i)
  document.write("<br>" + navigator.plugins[i].filename);
//-->
</script>
</body></html>
```

Das Beispiel klappert in einer `for`-Schleife alle installierten Plug-Ins ab und schreibt für jedes Plug-In eine Zeile mit dem Dateinamen des Plug-In. So werden am Ende die verantwortlichen

Dateien aller installierten Plug-Ins aufgelistet. Bei der Definition der `for`-Schleife dient die Eigenschaft `navigator.plugins.length` als Abbruchbedingung.

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.



Speichert den Produktnamen eines Plug-In. Das Beispiel klappert in einer `for`-Schleife alle installierten Plug-Ins ab und schreibt für jedes Plug-In eine Zeile mit dem Namen des Plug-In. So werden am Ende alle installierten Plug-Ins aufgelistet.

```
<html><head><title>Test</title>
</head><body>
<b>Folgende Plug-Ins sind installiert:</b>
<script type="text/javascript">
<!--
for(i=0; i<navigator.plugins.length; ++i)
  document.write("<br>" + navigator.plugins[i].name);
//-->
</script>
</body></html>
```

**Beachten Sie:** Der MS Internet Explorer 5.0 Macintosh Edition interpretiert dieses Beispiel auch.

## 20.29 Number

### Eigenschaften

|                   |                              |
|-------------------|------------------------------|
| MAX_VALUE         | (größte speicherbare Zahl)   |
| MIN_VALUE         | (kleinste speicherbare Zahl) |
| NaN               | (keine gültige Zahl)         |
| NEGATIVE_INFINITY | (Zahl zu klein)              |
| POSITIVE_INFINITY | (Zahl zu groß)               |

### Methoden

|                              |                    |
|------------------------------|--------------------|
| <code>toExponential()</code> | (Typkonvertierung) |
| <code>toFixed()</code>       | (Typkonvertierung) |
| <code>toPrecision()</code>   | (Typkonvertierung) |
| <code>toString()</code>      | (Typkonvertierung) |

### Number: Allgemeines zur Verwendung

Über das Objekt `Number` haben Sie Zugriff auf Eigenschaften numerischer Werte. So können Sie ermitteln, ob ein Wert eine gültige Zahl ist oder welches die maximale positive oder negative Zahl ist, die in einer numerischen Variablen gespeichert werden kann. Eigenschaften des `Number`-Objekts können Sie direkt ansprechen, indem Sie `Number` davor notieren.

Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var groessteZahl = Number.MAX_VALUE;
alert(groessteZahl);
var Monat = 13;
if (Monat < 1 || Monat > 12)
{
    Monat = Number.NaN;
    alert(Monat);
}
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert zunächst eine Variable namens `groessteZahl`. Dieser Variablen wird die Eigenschaft `Number.MAX_VALUE` zugewiesen. Anschließend steht in `groessteZahl` die größte Zahl, die JavaScript verarbeiten kann. Die Zahl wird zur Kontrolle ausgegeben. Da die Zahl viel zu groß ist, um sie direkt darzustellen, wird sie von JavaScript automatisch in Exponential-schreibweise dargestellt.

Dann definiert das Beispiel eine Variable namens `Monat` und weist dieser Variablen den Wert 13 zu. Danach wird abgefragt, ob der Wert von `Monat`, wie es sich für Monate gehört, zwischen 1 und 12 liegt. Wenn nicht (und das ist im Beispiel ja der Fall), wird der Variablen `Monat` die Eigenschaft `Number.NaN` zugewiesen. Der Wert von `Monat` ist danach `NaN`. Das bedeutet so viel wie: auf jeden Fall keine bzw. eine ungültige Zahl. Zur Kontrolle wird der neue Wert von `Monat` ausgegeben.



### MAX\_VALUE

Speichert die größte erlaubte Zahl. Das Beispiel definiert eine Variable namens `groessteZahl`. Dieser Variablen wird die Eigenschaft `Number.MAX_VALUE` zugewiesen. Anschließend steht in `groessteZahl` die größte Zahl, die JavaScript verarbeiten kann. Das Ergebnis wird zur Kontrolle ausgegeben.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var groessteZahl = Number.MAX_VALUE;
alert(groessteZahl);
//-->
</script>
</head><body>
</body></html>
```





**MIN\_VALUE**

Speichert die kleinste (negative) erlaubte Zahl. Das Beispiel definiert eine Variable namens `kleinsteZahl`. Dieser Variablen wird die Eigenschaft `Number.MIN_VALUE` zugewiesen. Anschließend steht in `kleinsteZahl` die »größte« negative Zahl, die JavaScript verarbeiten kann. Das Ergebnis wird zur Kontrolle ausgegeben.

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var kleinsteZahl = Number.MIN_VALUE;
  alert(kleinsteZahl);
//-->
</script>
</head><body>
</body></html>
```





**NaN**

NaN ist ein vordefinierter Wert (*keine gültige Zahl*). Dieser Wert kann z.B. als Rückgabewert für ungültige Zahlen verwendet werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahl_1 = "3a";
var Zahl_2 = "3";
if(Zahl_1 != 3) Zahl_1 = Number.NaN;
if(Zahl_2 != 3) Zahl_2 = Number.NaN;
alert(Zahl_1);
alert(Zahl_2);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert zwei Variablen `Zahl_1` und `Zahl_2`. Dabei wird `Zahl_1` mit dem Wert `3a` vorbelegt, `Zahl_2` mit dem Wert `3`. Anschließend wird für beide Variablen abgefragt, ob die Variable ungleich der Zahl `3` ist. Wenn ja, also wenn der Wert nicht `3` ist, wird der entsprechenden Variablen die Eigenschaft `Number.NaN` (*keine gültige Zahl*) zugewiesen. Abschließend werden beide Variableninhalte ausgegeben. `Zahl_1` hat nun den Wert `NaN`, weil es ungleich `3` war und deshalb die Eigenschaft `NaN` zugewiesen bekam. `Zahl_2` dagegen speichert den Wert `3`, da es als gültige Zahl `3` anerkannt wurde – obwohl sie bei der Initialisierung die `3` als Zeichenkette zugewiesen bekam: Hier greift die interne automatische Typumwandlung in JavaScript.

**Beachten Sie:** Mit der Eigenschaft `Number.NaN` können Sie nicht die Gültigkeit einer Zahl prüfen, sondern ihr nur die Eigenschaft *keine gültige Zahl* zuweisen. Die Gültigkeit einer Zahl können Sie mit der Methode `isNaN()` prüfen.

**NEGATIVE\_INFINITY**

Dies ist ein vordefinierter Wert (Ausgabewert: `-Infinity`). Er gibt an, ob eine (negative) Zahl zu klein ist für JavaScript, also außerhalb des Bereichs liegt, dessen Grenze durch `MIN_VALUE` markiert ist. Der Wert kann z.B. als Rückgabewert für zu kleine Zahlen verwendet werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  var Zahl = Number.MAX_VALUE * -2;
  alert(Zahl);
  var Monat=-1;
  if (Monat<1) Monat=Number.NEGATIVE_INFINITY;
  alert(Monat);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert eine Variable `Zahl`, der als Wert das Produkt aus `Number.MAX_VALUE`, also der größten möglichen Zahl, und `-2` zugewiesen wird. Die Multiplikation erzeugt eine negative Zahl, die kleiner ist als die kleinste erlaubte Zahl. Dadurch erhält die Variable `Zahl` den Wert `-Infinity`. Dieser bedeutet, dass der erlaubte Wertebereich unterschritten wurde. Im ersten Teil des Beispiels weist JavaScript automatisch dem Ergebnis den Wert von `Number.NEGATIVE_INFINITY` zu. Im zweiten Teil wird eine Variable `Monat` deklariert. Diese erhält den ungültigen Wert `-1`. In der nachfolgenden Abfrage wird geprüft, ob `Monat` kleiner als 1 ist, und ihr wird in diesem Fall die Eigenschaft `Number.NEGATIVE_INFINITY` zugewiesen.

**Beachten Sie:** Mit der Eigenschaft `Number.NEGATIVE_INFINITY` können Sie **nicht** prüfen, ob eine Zahl zu klein für JavaScript ist. Die Gültigkeit einer Zahl können Sie mit der Methode `isFinite()` prüfen.

**POSITIVE\_INFINITY**

Dies ist ein vordefinierter Wert (Ausgabewert: `Infinity`). Er gibt an, ob eine Zahl zu groß ist für JavaScript, also außerhalb des Bereichs liegt, dessen Grenze durch `MAX_VALUE` markiert ist. Der Wert kann z.B. als Rückgabewert für zu große Zahlen verwendet werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  Zahl = Number.MAX_VALUE * 2;
  alert(Zahl);
  var Monat=13;
  if (Monat>12) Monat=Number.POSITIVE_INFINITY;
  alert(Monat);
//-->
</script>
</head><body>
</body></html>
```



Das Beispiel definiert eine Variable `Zahl`, der als Wert das Produkt aus `Number.MAX_VALUE`, also der größten möglichen Zahl, und 2 zugewiesen wird. Die Multiplikation erzeugt eine Zahl, die größer ist als die größte erlaubte Zahl. Dadurch erhält die Variable `Zahl` den Wert `Infinity`. Dieser bedeutet, dass der erlaubte Wertebereich überschritten wurde. Im ersten Teil des Beispiels weist JavaScript dem Ergebnis den Wert von `Number.NEGATIVE_INFINITY` zu. Im zweiten Teil wird eine Variable `Monat` deklariert. Diese erhält den ungültigen Wert 13. In der nachfolgenden Abfrage wird geprüft, ob `Monat` größer als 12 ist. In diesem Fall wird ihr die Eigenschaft `Number.POSITIVE_INFINITY` zugewiesen.

**Beachten Sie:** Mit der Eigenschaft `Number.POSITIVE_INFINITY` können Sie **nicht** prüfen, ob eine Zahl zu groß für JavaScript ist. Die Gültigkeit einer Zahl können Sie mit der Methode `isFinite()` prüfen.

### ***toExponential()***

Erzwingt die Exponentialschreibweise einer Zahl. Das Beispiel definiert einen Wert und schreibt das Ergebnis der Anwendung von `toExponential()` auf diesen Wert ins Dokument. Geschrieben wird `1.3000012345e+1`.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Wert = 13.000012345;
document.write(Wert.toExponential());
//-->
</script>
</body></html>
```

### ***toFixed()***

Erzwingt eine bestimmte Anzahl Nachkommastellen. Das Beispiel definiert einen Wert und schreibt das Ergebnis der Anwendung von `toFixed(2)` auf diesen Wert ins Dokument. Geschrieben wird `7123.20`, weil der an die Methode im Beispiel übergebene Wert 2 so viel bedeutet wie: schneide nach 2 Nachkommastellen ab.

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Wert = 7123.203848;
document.write(Wert.toFixed(2));
//-->
</script>
</body></html>
```

**Beachten Sie:** Diese Methode rundet automatisch kaufmännisch. Würde der Wert im Beispiel `7123.209848` lauten, dann würde `7123.21` ermittelt. Um diese Automatik zu verhindern, können Sie `toPrecision()` verwenden.



Erzwingt eine bestimmte Genauigkeit bei der Darstellung einer Zahl. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
Wert = 7123.209;
document.write("precision(1) = " + Wert.toPrecision(1) + "<br>");
document.write("precision(2) = " + Wert.toPrecision(2) + "<br>");
document.write("precision(3) = " + Wert.toPrecision(3) + "<br>");
document.write("precision(4) = " + Wert.toPrecision(4) + "<br>");
document.write("precision(5) = " + Wert.toPrecision(5) + "<br>");
document.write("precision(6) = " + Wert.toPrecision(6) + "<br>");
document.write("precision(7) = " + Wert.toPrecision(7) + "<br>");
document.write("precision(8) = " + Wert.toPrecision(8) + "<br>");
//-->
</script>
</body></html>
```

Das Beispiel definiert einen Wert und schreibt dann mehrfach das Ergebnis der unterschiedlichen Anwendung von `toPrecision(x)` auf diesen Wert ins Dokument. Geschrieben wird:

```
precision(1) = 7e+3
precision(2) = 7.1e+3
precision(3) = 7.12e+3
precision(4) = 7123
precision(5) = 7123.2
precision(6) = 7123.21
precision(7) = 7123.209
precision(8) = 7123.2090
```



Wandelt eine Zahl in eine Zeichenkette (String) um. Diese Methode ist dann notwendig, wenn Sie Methoden des String-Objekts auf eine Zahl anwenden wollen. Die Methode `toString()` besitzt einen optionalen Parameter. Der Parameter gibt die Basis des gewünschten Zahlensystems an (z.B. 10 für das Dezimalsystem, oder 16 für das Hexadezimalsystem). Der optionale Parameter kann Werte zwischen 2 und 36 annehmen. Wird kein Parameter angegeben, so wird die Zahl als Dezimalzahl behandelt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Dezimalzahl = 15.5;
var Dualzahl=Dezimalzahl.toString(2);
alert(Dualzahl);

//-->
</script>
</body></html>
```

Das Beispiel definiert eine Variable namens `Dezimalzahl`. Diese Variable enthält die Dezimalzahl 15.5. Anschließend wird der Variablen `Dualzahl` der Rückgabewert der Methode `toString()` mit dem Parameter 2 zugewiesen. In der Variablen `Dualzahl` ist jetzt die Zahl 15.5 als Dualzahl-Zeichenkette gespeichert. Im Beispiel wird im Meldungsfenster 1111.1 ausgegeben. Damit fungiert die Methode `toString()` gleichzeitig als Umrechner von Zahlen des dezimalen Zahlensystems in Zahlen anderer Zahlensysteme. Mit der Methode `parseInt()` können Sie umgekehrt Zeichenketten von Zahlen anderer Zahlensysteme in das dezimale Zahlensystem konvertieren.

## 20.30 RegExp

### Eigenschaften

`$[1..9]` (geklammerte Unterausdrücke)

### Methoden

`exec()` (regulären Ausdruck anwenden)  
`test()` (regulären Ausdruck probenhalber anwenden)

### RegExp: Allgemeines zur Verwendung

Reguläre Ausdrücke dienen dazu, Suchausdrücke zu formulieren, um in Zeichenketten danach zu suchen und um gefundene Stellen durch andere zu ersetzen. Reguläre Ausdrücke können Sie in JavaScript direkt innerhalb entsprechender Methoden des `string`-Objekts anwenden, nämlich in den Methoden:

- `match()`
- `replace()`
- `search()`

Dort wird beschrieben, wie und wo Sie den regulären Ausdruck genau verwenden können, um Zeichenketten zu durchsuchen und Teile darin zu ersetzen. Das `RegExp`-Objekt von JavaScript brauchen Sie dagegen nur, wenn Sie reguläre Ausdrücke zur Laufzeit des Scripts dynamisch erzeugen und ändern wollen. Dazu können Sie eine Instanz eines `RegExp`-Objekts definieren. Auf diese Instanz können Sie anschließend die Eigenschaften und Methoden des `RegExp`-Objekts anwenden, die hier beschrieben werden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Ausgabe(Wert)
{
    var Ausdruck = /(\w+)\s(\w+)/;
    Ausdruck.exec(Wert);
    alert(RegExp.$2 + ", " + RegExp.$1);
}
//-->
```

```

</script>
</head><body>
<form action="">
<input name="User" value="Vorname Zuname">
<input type="button" value="Ausgabe" onClick="Ausgabe(this.form.User.value)">
</form>
</body></html>

```

Das Beispiel zeigt, wie sich ein Name des Schemas »Vorname Name« unter Verwendung eines regulären Ausdrucks in das Schema »Name, Vorname« verwandeln lässt. Das Beispiel enthält ein Formular mit einem Eingabefeld und einem Button. In dem Eingabefeld soll der Anwender seinen Namen in der Form »Vorname Name« eingeben. Deshalb ist das Feld auch schon so vorgelegt. Beim Anklicken des Buttons wird die Funktion `Ausgabe()` aufgerufen, die im Dateikopf definiert ist. Als Parameter bekommt die Funktion den Inhalt des Eingabefeldes übergeben.

Innerhalb der Funktion wird ein regulärer Ausdruck (Suchausdruck) definiert, der in der Variablen `Ausdruck` gespeichert wird. Solche regulären Ausdrücke werden in einfache Schrägstriche `/.../` eingeschlossen. Der reguläre Ausdruck im obigen Beispiel ist so zu lesen: Suche nach einer beliebigen Anzahl von aneinander gereihten Buchstaben ohne Ziffern und Satzzeichen und merke dir das Ergebnis = `(\w.+)`. Suche dann nach einem einzelnen Leerzeichen = `\s`. Suche dann noch mal nach einer beliebigen Buchstabenreihenfolge und merke dir das Ergebnis = `(\w.+)`. Auf diese Weise wird das Schema »Vorname Name« gefunden. Die beiden Klammern um das `(\w.+)` führen dazu, dass die beiden einzelnen Buchstabenreihenfolgen als einzelne Teile intern gemerkt werden. Später sind diese einzelnen gemerkten Teile separat ansprechbar.

Im Beispiel wird mit dem nächsten Befehl die eine wichtige Methode des `RegExp`-Objekts auf den zuvor definierten regulären Ausdruck angewendet, nämlich die Methode `exec()`. Dadurch wird der Ausdruck »ausgeführt« und direkt dem `RegExp`-Objekt zugewiesen. Im Beispiel wird mit `RegExp.$2` direkt auf den zweiten, zuvor intern gemerkten Klammerausdruck zugegriffen, und mit `RegExp.$1` auf den ersten dieser Ausdrücke. Durch die Anordnung mit dem Komma dazwischen bei der Ausgabe mit `alert()` erscheint der zuvor eingegebene »Vorname, Name« dann in dem anderen Schema »Name, Vorname«. Die folgende Übersicht zeigt, aus welchen Bestandteilen Sie einen regulären Ausdruck zusammensetzen können.

| Bestandteil     | Beispiel             | Beschreibung                                                                                                                       |
|-----------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------|
|                 | <code>/aus/</code>   | Findet »aus«, und zwar in »aus«, »Haus«, »auserlesen« und »Banause«.                                                               |
| <code>^</code>  | <code>/^aus/</code>  | Findet »aus« am Anfang des zu durchsuchenden Wertes, also in »aus« und »auserlesen«, sofern das die ersten Wörter im Wert sind.    |
| <code>\$</code> | <code>/aus\$/</code> | Findet »aus« am Ende des zu durchsuchenden Wertes, also in »aus« und »Haus«, sofern das die letzten Wörter im Wert sind.           |
| <code>*</code>  | <code>/aus*/</code>  | Findet »aus«, »auss« und »aussssss«, also das letzte Zeichen vor dem Stern 0 oder beliebig oft hintereinander wiederholt.          |
| <code>+</code>  | <code>/aus+/</code>  | Findet »auss« und »aussssss«, also das letzte Zeichen vor dem Stern mindestens einmal oder beliebig oft hintereinander wiederholt. |
| <code>.</code>  | <code>/ .aus/</code> | Findet »Haus« und »Maus«, also ein beliebiges Zeichen an einer bestimmten Stelle.                                                  |

| Bestandteil | Beispiel  | Beschreibung                                                                                                                                                                     |
|-------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .+          | /.+aus/   | Findet »Haus« und »Kehraus«, also eine beliebige Zeichenfolge an einer bestimmten Stelle. Zusammensetzung aus beliebiges Zeichen und beliebig viele davon.                       |
| \b          | /\baus\b/ | Findet »aus« als einzelnes Wort. \b bedeutet eine Wortgrenze.                                                                                                                    |
| \B          | /\Baus\B/ | Findet »aus« nur innerhalb von Wörtern, z.B. in »hausen« oder »Totalausfall«. \B bedeutet keine Wortgrenze.                                                                      |
| \d          | /\d.+\B/  | Findet eine beliebige ganze Zahl. \d bedeutet eine Ziffer (0 bis 9).                                                                                                             |
| \D          | /\D.+/    | Findet »-fach« in »3-fach«, also keine Ziffer.                                                                                                                                   |
| \f          | /\f/      | Findet ein Seitenvorschubzeichen.                                                                                                                                                |
| \n          | /\n/      | Findet ein Zeilenvorschubzeichen.                                                                                                                                                |
| \r          | /\r/      | Findet ein Wagenrücklaufzeichen.                                                                                                                                                 |
| \t          | /\t/      | Findet ein Tabulatorzeichen.                                                                                                                                                     |
| \v          | /\v/      | Findet ein vertikales Tabulatorzeichen.                                                                                                                                          |
| \s          | /\s/      | Findet jede Art von white space, also \f\n\t\v und Leerzeichen.                                                                                                                  |
| \S          | /\S.+/    | Findet ein beliebiges einzelnes Zeichen, das kein white space ist, also kein \f\n\t\v und kein Leerzeichen.                                                                      |
| \w          | /\w.+/    | Findet alle alphanumerischen Zeichen und den Unterstrich (typische Bedingung etwa für programmiersprachengerechte selbst vergebene Namen).                                       |
| \W          | /\W/      | Findet ein Zeichen, das nicht alphanumerisch und auch kein Unterstrich ist (typisch zum Suchen nach illegalen Zeichen bei programmiersprachengerechten selbst vergebenen Namen). |
| \(          | /(aus)/   | Findet »aus« und merkt es sich intern. Bis zu 9 solcher Klammern (Merkplätze) sind in einem regulären Ausdruck erlaubt.                                                          |
| /.../g      | /aus/g    | Findet »aus« so oft, wie es in dem gesamten zu durchsuchenden Bereich vorkommt. Die Fundstellen werden intern in einem Array gespeichert.                                        |
| /.../i      | /aus/i    | Findet »aus«, »Aus« und »AUS«, also unabhängig von Groß-/Kleinschreibung.                                                                                                        |
| /.../gi     | /aus/gi   | Findet »aus«, so oft, wie es in dem gesamten zu durchsuchenden Bereich vorkommt (g) und unabhängig von Groß-/Kleinschreibung (i).                                                |



**\$[1..9]**

Speichert gemerkte Bestandteile eines regulären Ausdrucks, also geklammerte Teile. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Vollversion = /(\d)/;
Vollversion.exec(navigator.userAgent);
document.write("Volle Versionsnummer Ihres Browsers: " + RegExp.$1);
//-->
</script>
</body></html>
```

Das Beispiel definiert einen regulären Ausdruck namens `Vollversion`, der nach dem ersten Vorkommen einer Ziffer sucht und sich die Fundstelle durch eine Klammer `= (\d)` merkt.

Dieser reguläre Ausdruck wird dann auf die Objekteigenschaft `navigator.userAgent` angewendet, die Angaben zum Browser des Anwenders speichert. Da dieser Wert einem bestimmten Schema folgt, ist klar, dass die erste darin vorkommende Ziffer die Vollversion des Browsers darstellt (sofern sie einstellig ist). Im Beispiel wird die gefundene Versionsnummer dynamisch in die Datei geschrieben. Dabei wird direkt über das `RegExp`-Objekt auf die Eigenschaft `$1` zugegriffen. Das ist der erste in einer Klammer gemerkte Teilausdruck des aktuellen regulären Ausdrucks, im Beispiel die Versionsziffer.



Wendet einen regulären Ausdruck auf eine Zeichenkette an. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var derSatz = "Auf der Mauer";
  var Suche = /(au)/g;
  var Ergebnis = Suche.exec(derSatz);
  document.write(Ergebnis.length + " Suchtreffer");
  //-->
</script>
</body></html>
```

Das Beispiel demonstriert den Zusammenhang: Ein regulärer Ausdruck wird definiert, im Beispiel ein regulärer Ausdruck mit dem Namen `Suche`. Mit einer Anweisung wie `Suche.exec()` können Sie dann die Suche starten. Als Parameter wird der Methode in der Regel der zu durchsuchende Ausdruck übergeben. Die Methode gibt die gefundenen Suchtreffer zurück. Die Variable, in der der Rückgabewert gespeichert wird (im Beispiel die Variable `Ergebnis`), wird automatisch zu einem Array, wenn mehr als ein Suchtreffer gefunden wurde. Im Beispiel werden zwei Treffer gefunden. Deshalb lässt sich auch die Array-Eigenschaft `length` auf die Variable `Ergebnis` anwenden.



Testet vorab, ob ein regulärer Ausdruck zu Suchtreffern führt oder nicht. Gibt `true` zurück, wenn etwas gefunden wurde, und `false`, wenn nichts gefunden wurde. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var derSatz = "Auf der Mauer";
  var Suche = /(sonstwas)/g;
  var Ergebnis = Suche.test(derSatz);
  if(Ergebnis == false)
    document.write("nichts gefunden");
  //-->
</script>
</body></html>
```

Das Beispiel definiert einen regulären Ausdruck namens `Suche` und wendet ihn mit Hilfe der Methode `test()` auf die Variable `derSatz` an. Der Rückgabewert wird in der Variablen `Ergebnis` gespeichert. Wenn nichts gefunden wird, wird ein entsprechender Hinweis geschrieben. Im Beispiel ist das der Fall.

## 20.31 Screen

### Eigenschaften

|                          |                     |
|--------------------------|---------------------|
| <code>availHeight</code> | (verfügbare Höhe)   |
| <code>availWidth</code>  | (verfügbare Breite) |
| <code>colorDepth</code>  | (Farbtiefe)         |
| <code>height</code>      | (Höhe)              |
| <code>pixelDepth</code>  | (Farbauflösung)     |
| <code>width</code>       | (Breite)            |

### *screen: Allgemeines zur Verwendung*

Mit dem Objekt `screen` können Sie Angaben zum Bildschirm des Anwenders ermitteln. Diese Angaben zu kennen ist wichtig, wenn Sie professionelles »Screendesign« mit HTML und JavaScript betreiben wollen. Das `screen`-Objekt ist bei Netscape und beim MS Internet Explorer ab den 4er Versionen verfügbar. Eigenschaften des `screen`-Objekts können Sie direkt ansprechen, indem Sie `screen` davor notieren. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
  alert(screen.width + "x" + screen.height);
//-->
</script>
</head><body>
</body>
</html>
```

Das Beispiel gibt beim Einlesen der Datei Breite (`screen.width`) mal Höhe (`screen.height`) des Anwenderbildschirms aus. Es handelt sich dabei um die Angabe in Pixeln, Ergebnisse sind also etwa 800x600 oder 1024x768.



### *availHeight*

Speichert die maximal verfügbare Bildschirmhöhe in Pixeln, die eine Anwendung im Vollbildmodus einnehmen kann. Unterscheidet sich von `screen.height`, wenn feste Bildelemente wie immer eingeblendete Taskleisten usw. einen Teil des Bildschirms oben oder unten einnehmen. Während `height` dann beispielsweise 768 liefert, gibt `availHeight` z.B. nur 712 aus, weil eine Taskleiste 56 Pixel Höhe einnimmt. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
```

```

if(screen.availHeight != screen.height)
  alert("Sie sehen mehr von dieser Seite, wenn Sie die Taskleiste ausblenden");
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel vergleicht die beiden Eigenschaften `screen.availHeight` und `screen.height`. Wenn beide nicht den gleichen Wert speichern, hat der Anwender vermutlich seine Taskleiste immer im Vordergrund. Das Beispiel gibt in diesem Fall einen entsprechenden Hinweis aus.



### ***availWidth***

Speichert die maximal verfügbare Bildschirmbreite in Pixeln, die eine Anwendung im Vollbildmodus einnehmen kann. Unterscheidet sich von `screen.width`, wenn feste Bildelemente wie immer eingeblendete Taskleisten usw. links oder rechts einen Teil des Bildschirms einnehmen. Während `width` dann beispielsweise 1024 liefert, gibt `availWidth` z.B. nur 940 aus, weil an einer Seite eine Taskleiste 84 Pixel Breite einnimmt. Ein Beispiel:

```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(screen.availWidth != screen.width)
  alert("Sie haben wohl seitlich eine Taskbar!");
//-->
</script>
</head><body>
</body></html>

```

Das Beispiel vergleicht die beiden Eigenschaften `screen.availWidth` und `screen.width`. Wenn beide nicht den gleichen Wert speichern, hat der Anwender vermutlich seine Taskleiste immer im Vordergrund. Das Beispiel gibt in diesem Fall einen entsprechenden Hinweis aus.



### ***colorDepth***

Speichert die Anzahl Bits, die der Anwenderbildschirm zur Darstellung einer Farbe an jedem Pixelpunkt des Bildschirms verwendet. Ein Wert von 16 bedeutet beispielsweise: Der Bildschirm kann  $2 \text{ hoch } 16$ , also 65536 Farben darstellen. Diese Eigenschaft hat dann einen Wert, wenn der Anwenderbildschirm intern eine Farbpalette benutzt. Andernfalls wird `undefined` (Netscape) oder `null` (MS Internet Explorer) gespeichert. Bei Netscape können Sie dann die Eigenschaft `pixelDepth` abfragen. Ein Beispiel:

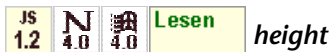
```

<html><head><title>Test</title>
<script type="text/javascript">
<!--
alert(Math.pow(2,screen.colorDepth));
//-->
</script>
</head><body>
</body></html>

```



Das Beispiel ermittelt die Anzahl Farben, die der Anwenderbildschirm darstellen kann, durch Anwendung der Math-Methode `pow()`. Mit `Math.pow(2,screen.colorDepth)` wird der Wert, den `screen.colorDepth` speichert, als Exponent zur Basis 2 benutzt. Das Ergebnis ist also 2 hoch `screen.colorDepth`.



Speichert die absolut verfügbare Höhe des Anwenderbildschirms in Pixeln, besser bekannt als Höhe der Bildschirmauflösung. Typische Werte sind 480, 600 und 768. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(screen.height >= 768)
  window.location.href = "datei1.htm"
else
  window.location.href = "datei2.htm"
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel fragt ab, ob die verfügbare Bildschirmhöhe größer oder gleich 768 ist. Wenn ja, wird automatisch `datei1.htm` geladen, wenn nicht (also wenn die Bildschirmhöhe kleiner ist), wird `datei2.htm` geladen.



Speichert die Farbtiefe des Anwenderbildschirms in Bit pro Pixel. Diese Eigenschaft hat dann einen Wert, wenn der Anwenderbildschirm intern eine Farbpalette benutzt. Andernfalls wird `undefined` (Netscape) oder `null` (MS Internet Explorer) gespeichert, und Sie müssen die Eigenschaft `colorDepth` abfragen. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JavaScript" type="text/javascript">
<!--
if(screen.pixelDepth)
  alert(screen.pixelDepth);
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel ermittelt, ob für `screen.pixelDepth` ein Wert verfügbar ist. Wenn ja, wird der Wert in einem Meldungsfenster ausgegeben.



Speichert die absolut verfügbare Breite des Anwenderbildschirms in Pixeln, besser bekannt als Breite der Bildschirmauflösung. Typische Werte sind 640, 800 und 1024. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
if(screen.width >= 1024)
  window.location.href = "datei1.htm"
else
  window.location.href = "datei2.htm"
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel fragt ab, ob die verfügbare Bildschirmbreite größer oder gleich 1024 ist. Wenn ja, wird automatisch *datei1.htm* geladen, wenn nicht (also wenn die Bildschirmbreite kleiner ist), wird *datei2.htm* geladen.

## 20.32 string

### Eigenschaften

length (Anzahl Zeichen)

### Methoden

|                |                                            |
|----------------|--------------------------------------------|
| anchor()       | (Verweisanker erzeugen)                    |
| big()          | (großen Text erzeugen)                     |
| blink()        | (blinkenden Text erzeugen)                 |
| bold()         | (fetten Text erzeugen)                     |
| charAt()       | (Zeichen an einer Position ermitteln)      |
| charCodeAt()   | (Latin-1-Codewert an einer Position)       |
| concat()       | (Zeichenketten zusammenfügen)              |
| fixed()        | (Teletyper-Text erzeugen)                  |
| fontcolor()    | (Schriftfarbe erzeugen)                    |
| fontSize()     | (Schriftgröße erzeugen)                    |
| fromCharCode() | (Latin-1-Zeichenkette erzeugen)            |
| indexOf()      | (Position eines Zeichens ermitteln)        |
| italics()      | (kursiven Text erzeugen)                   |
| lastIndexOf()  | (letzte Position eines Zeichens ermitteln) |
| link()         | (Verweis erzeugen)                         |
| match()        | (regulären Ausdruck anwenden)              |
| replace()      | (regulären Ausdruck anwenden und ersetzen) |
| search()       | (Suchen mit regulärem Ausdruck)            |
| slice()        | (Teil aus Zeichenkette extrahieren)        |

|                            |                                          |
|----------------------------|------------------------------------------|
| <code>small()</code>       | (kleinen Text erzeugen)                  |
| <code>split()</code>       | (Zeichenkette aufsplitten)               |
| <code>strike()</code>      | (durchgestrichenen Text erzeugen)        |
| <code>sub()</code>         | (tiefgestellten Text erzeugen)           |
| <code>substr()</code>      | (Teilzeichenkette ab Position ermitteln) |
| <code>substring()</code>   | (Teilzeichenkette ermitteln)             |
| <code>sup()</code>         | (hochgestellten Text erzeugen)           |
| <code>toLowerCase()</code> | (alles klein schreiben)                  |
| <code>toUpperCase()</code> | (alles groß schreiben)                   |

### Allgemeines zur Verwendung

Die Eigenschaften und Methoden des Objektes `string` können Sie auf alle Zeichenketten anwenden. So können Sie in einer Zeichenkette z.B. alle darin enthaltenen Kleinbuchstaben in Großbuchstaben umwandeln oder HTML-Formatierungen auf die Zeichenkette anwenden. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Ort = "Friedrichshafen"
alert(Ort + " hat " + Ort.length + " Buchstaben")
//-->
</script>
</head>
<body>
</body></html>
```

Im Beispiel wird eine Variable `Ort` definiert, der ein Wert zugewiesen wird, nämlich der Wert Friedrichshafen. Anschließend wird in einem Meldungsfenster ausgegeben: Friedrichshafen hat 15 Buchstaben. Dies geschieht durch einen Aufruf der Objekteigenschaft `length`. Vor dem Punkt kann also eine Variable oder auch irgendeine andere JavaScript-Objekteigenschaft stehen, in der eine Zeichenkette gespeichert ist. Hinter dem Punkt folgt die gewünschte Eigenschaft oder Methode des `string`-Objekts. Zeichenketten in JavaScript werden als eine Kette von Einzelzeichen betrachtet. Jedes Einzelzeichen hat seine Position in der Zeichenkette. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Name = "Hans";
alert(Name[0] + Name[1] + Name[2] + Name[3]);
//-->
</script>
</head>
<body>
</body></html>
```

Das Beispiel, das in dieser Form allerdings nur mit Netscape, nicht mit dem MS Internet Explorer und Opera 5.12 funktioniert, definiert eine Variable mit dem Namen `Name`. Diese Variable erhält den Wert "Hans". In einem Meldungsfenster wird anschließend zeichenweise dieser Wert der Variablen `Name` ausgegeben. Das Ergebnis ist das gleiche, wie wenn einfach `alert(Name)`

notiert würde. Aber an dieser Notationsform können Sie erkennen, wie Zeichenketten funktionieren: `Name[0]` ist das H, `Name[1]` das a, `Name[2]` das n und `Name[3]` das s. Es wird also immer bei 0 zu zählen begonnen.

### Lesen *length*

Speichert die Anzahl Zeichen einer Zeichenkette, bei der Zeichenkette »Guten Abend« beispielsweise 11. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
var Eingabe = window.prompt("Irgendetwas eingeben","");
alert("Eingegebene Zeichen: " + Eingabe.length);
//-->
</script></body></html>
```

Das Beispiel blendet ein Dialogfenster ein, in dem der Anwender irgendetwas eingeben kann. Der eingegebene Wert wird in der Variablen `Eingabe` gespeichert. Nach Beenden des Dialogfensters wird ein Meldungsfenster ausgegeben, das mit `Eingabe.length` ausgibt, wie viele Zeichen eingegeben wurden.

### *anchor()*

Formatiert eine Zeichenkette als Verweisanker, der mit Hilfe eines HTML-Verweises angesprungen werden kann. Hat also die gleiche Wirkung wie das HTML-Konstrukt `<a name=...></a>`. Als Parameter erwartet die Methode den gewünschten Ankernamen. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
var Anker = "Inhaltsverzeichnis";
document.write(Anker.anchor("a1"));
for(var i=1;i<100;++i)
  document.write("<br>Zeile " + i);
document.write("<br><a href=\"\#Anker\">Testverweis</a>");
//-->
</script></body></html>
```

Das Beispiel definiert eine Variable `Anker` mit dem Wert `Inhaltsverzeichnis`. Mit Hilfe von `document.write()` wird daraus ein Verweisanker geschrieben. Die Konstruktion `Anker.anchor("a1")` bewirkt dabei das Gleiche wie das HTML-Konstrukt `<a name="a1">Inhaltsverzeichnis</a>`. Zu Testzwecken werden im Beispiel anschließend 100 Zeilen in die Datei geschrieben und dahinter ein Verweis. Beim Klicken auf den Verweis wird der mit JavaScript erzeugte Verweisanker angesprungen.

**Beachten Sie:** Das Beispiel funktioniert nur eingeschränkt. Opera 5.12 und Netscape 4.x springen nach dem Klick auf den Verweis an den Anfang des Dokumentes. Netscape 6 und der MS Internet Explorer ignorieren den Anker vollständig. Tests ergaben jedoch, dass alle Browser den String erzeugen und auch einen Eintrag im Objekt `anchors` anlegen.



Formatiert eine Zeichenkette mit größerer Schrift, genau wie die HTML-Formatierung `<big>...</big>`. Erwartet als Parameter eine Zeichenkette. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var update = document.lastModified;
  document.write(update.big());
//-->
</script></body></html>
```

Im Beispiel ermittelt ein JavaScript mit Hilfe von `document.lastModified`, wann die Datei zuletzt geändert wurde. In der Variablen `update` steht dann eine entsprechende Zeichenkette. Mit Hilfe von `document.write()` wird der Wert von `update` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<big>...</big>`.



Formatiert eine Zeichenkette als blinkenden Text, genau wie die HTML-Formatierung `<blink>...</blink>`. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Ausgabe = "Best view only with your " + navigator.appName";
  document.write(Ausgabe.blink());
//-->
</script></body></html>
```

Im Beispiel wird eine Variable namens `Ausgabe` mit einem Wert belegt. Mit Hilfe von `document.write()` wird der Wert von `Ausgabe` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<blink>...</blink>`.

**Beachten Sie:** Obwohl weder Opera 5.12, Netscape 6 noch der MS Internet Explorer die HTML-Notation `<blink>...</blink>` unterstützen, interpretieren diese Browser jedoch die Methode und erzeugen den entsprechenden HTML-Code.



Formatiert eine Zeichenkette als fetten Text, genau wie die HTML-Formatierung `<b>...</b>`. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var DieseSeite = window.location.href;
  document.write("Adresse: " + DieseSeite.bold());
//-->
</script></body></html>
```

Im Beispiel ermittelt ein JavaScript mit Hilfe von `window.location.href` den URI der aktuellen Datei. In der Variablen `DieseSeite` steht dann eine entsprechende Zeichenkette. Mit Hilfe von `document.write()` wird der Wert von `update` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<b>...</b>`.

### **charAt()**

Liefert dasjenige Zeichen zurück, das in einer Zeichenkette an einer bestimmten Position steht. Erwartet als Parameter die gewünschte Position. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Vollversion = navigator.appVersion.charAt(0);
  alert(Vollversion);
//-->
</script></body></html>
```

Im Beispiel ermittelt ein JavaScript mit Hilfe von `navigator.appVersion`, welche Version seines Browsers der Anwender benutzt. Auf `navigator.appVersion` wird jedoch zusätzlich die Methode `charAt()` angewendet. Als Parameter wird 0 übergeben, denn es soll das erste Zeichen des Wertes ermittelt werden, der in `navigator.appVersion` steht (dies ist immer die Nummer der Vollversion, also bei Netscape 4.01 etwa die 4). Das Ergebnis der Konstruktion wird in der Variablen `Vollversion` gespeichert. Zur Kontrolle wird der Wert dieser Variablen anschließend in einem Meldungsfenster ausgegeben.

### **charCodeAt()**

Liefert den Latin-1-Zeichensatzwert eines Zeichens zurück, das in einer Zeichenkette an einer bestimmten Position steht. Erwartet als Parameter die gewünschte Position. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Name = "Hans";
  alert(Name.charCodeAt(1));
//-->
</script></body></html>
```

Im Beispiel wird eine Variable `Name` mit dem Wert `Hans` belegt. In einem Meldungsfenster wird anschließend ausgegeben, welchen Zeichensatzwert das zweite Zeichen dieser Variablen hat, also das »a« in »Hans«. Dazu wird `charCodeAt` als Parameter 1 übergeben, denn in JavaScript hat das erste Zeichen einer Zeichenkette immer die Position 0, das zweite die Position 1 usw. Als Ergebnis wird 97 angezeigt. Dies ist der Zeichensatzwert des kleinen »a«.

### **concat()**

Hängt eine Zeichenkette an eine andere Zeichenkette an. Erwartet als Parameter die Zeichenkette, die an die erste angehängt werden soll. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Vorname = "Hans ";
  var Zuname = "Hansen";
  var Name = Vorname.concat(Zuname);
  alert(Name);
//-->
</script></body></html>
```

Im Beispiel werden zwei Variablen `Vorname` und `Zuname` mit entsprechenden Werten belegt. Dann wird die Methode `concat()` auf die Variable `Vorname` angewendet. Als Parameter wird der Methode die Variable `Zuname` übergeben. Auf diese Weise wird `Zuname` an `Vorname` angehängt. Die Methode `concat()` gibt die zusammengefügte Zeichenkette zurück. Im Beispiel wird dieser Rückgabewert in der Variablen `Name` gespeichert. Zur Kontrolle wird diese Variable anschließend in einem Meldungsfenster ausgegeben.

#### **fixed()**

Formatiert eine Zeichenkette als Teletyper-Text, genau wie die HTML-Formatierung `<tt>...</tt>`. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var update = document.lastModified;
  document.write("zuletzt aktualisiert: " + update.fixed());
//-->
</script></body></html>
```

Im Beispiel ermittelt ein JavaScript mit Hilfe von `document.lastModified`, wann die Datei zuletzt geändert wurde. In der Variablen `update` steht dann eine entsprechende Zeichenkette. Mit Hilfe von `document.write()` wird der Wert von `update` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<tt>...</tt>`.

#### **fontcolor()**

Formatiert eine Zeichenkette in farbiger Schrift, genau wie die HTML-Formatierung `<font color=>...</font>`. Erwartet als Parameter die gewünschte Farbe. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Zeit = new Date;
  var Datum = Zeit.getDate()+"."+ (Zeit.getMonth()+1)+"."+Zeit.getFullYear();
  document.write(Datum.fontcolor("#FF0000"));
//-->
</script></body></html>
```

Das Beispiel erzeugt mit Hilfe des `Date`-Objekts das aktuelle Tagesdatum und speichert dieses in der Variablen `Datum`. Mit Hilfe von `document.write()` wird der Wert von `Datum` in die Datei

geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<font color="#FF0000">...</font>`. Als Werte sind hexadezimale Werte oder Farbnamen erlaubt (siehe Kapitel 3.5, *Farben definieren in HTML*).

### **fontSize()**

Formatiert eine Zeichenkette in einer relativen Schriftgröße zwischen 1 und 7, wobei 3 die Normalschriftgröße ist. Hat die gleiche Wirkung wie die HTML-Formatierung `<font size=>...</font>`. Erwartet als Parameter die gewünschte Größe der Schrift. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Ausgabe = "Meine Homepage";
  document.write(Ausgabe.fontSize(7));
//-->
</script></body></html>
```

Im Beispiel wird eine Variable namens `Ausgabe` mit einem Wert belegt. Mit Hilfe von `document.write()` wird der Wert von `Ausgabe` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<font size="7">...</font>`.

### **fromCharCode()**

Interpretiert eine Reihe von numerischen Parametern, die als Parameter übergeben werden, als Latin-1-Zeichensatzwerte. Erzeugt eine entsprechende Zeichenkette daraus. Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Test = String.fromCharCode(65,66,67);
  alert(Test);
//-->
</script></body></html>
```

Im Beispiel wird mit `String.fromCharCode()` die Zeichenkette »ABC« erzeugt und in der Variablen `Test` gespeichert. Die Zeichenkette kommt dadurch zustande, dass 66 der dezimale Zeichensatzwert für A ist, 67 derjenige für B und 68 der für C. Im Beispiel wird das Ergebnis zur Kontrolle in einem Meldungsfenster ausgegeben. Wichtig ist, vor der Methode `fromCharCode()` immer das feste Schlüsselwort `String` (großgeschrieben) zu notieren.

### **indexOf()**

Ermittelt das erste Vorkommen eines Zeichens oder einer Zeichenkette innerhalb einer Zeichenkette und gibt zurück, an wievielter Stelle das Zeichen in der Zeichenkette steht. Die Zählung beginnt bei 0. Wenn die Suche erfolglos ist, wird -1 zurückgegeben. Optional ist es möglich, die Funktion in einem zweiten Parameter anzuweisen, ab der wievielten Stelle in der Zeichenkette sie mit der Suche beginnen soll. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
```



```

var Aussage = "Der Mensch ist dem Mensch sein Feind";
var Suche = Aussage.indexOf("Mensch");
alert("gefunden bei Position: " + Suche);
//-->
</script></body></html>

```

Das Beispiel belegt eine Variable `Aussage` mit einem Wert. Anschließend wird der Wert dieser Variablen mit `indexOf()` nach dem ersten Vorkommen von `Mensch` innerhalb der Zeichenkette gesucht. In der Variablen `Suche` wird der Rückgabewert von `indexOf()` gespeichert. Im Beispiel wird 4 zurückgegeben, da die Zeichenfolge `Mensch` in der durchsuchten Zeichenkette zum ersten Mal an 5. Stelle der Zeichenkette beginnt. Vergleiche auch: `lastIndexOf()`.

### ***italics()***

Formatiert eine Zeichenkette kursiv, genau wie die HTML-Formatierung `<i>...</i>`. Ein Beispiel:

```

<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
var Zeit = new Date;
var Uhrzeit = Zeit.getHours()+":"+Zeit.getMinutes();
document.write(Uhrzeit.italics());
//-->
</script></body></html>

```

Das Beispiel erzeugt mit Hilfe des `Date`-Objekts die aktuelle Uhrzeit und speichert diese in der Variablen `Uhrzeit`. Mit Hilfe von `document.write()` wird der Wert von `Uhrzeit` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<i>...</i>`.

### ***lastIndexOf()***

Ermittelt das letzte Vorkommen eines Zeichens oder einer Zeichenkette innerhalb einer Zeichenkette und gibt zurück, an wievielter Stelle das Zeichen in der Zeichenkette steht. Die Zählung beginnt bei 0. Wenn die Suche erfolglos ist, wird -1 zurückgegeben. Optional ist es möglich, die Funktion in einem zweiten Parameter anzuweisen, ab der wievielten Stelle in der Zeichenkette sie mit der Suche beginnen soll. Ein Beispiel:

```

<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
var Aussage = "Der Mensch ist dem Mensch sein Feind";
var Suche = Aussage.lastIndexOf("Mensch");
alert("gefunden bei Position: " + Suche);
//-->
</script></body></html>

```

Das Beispiel belegt eine Variable `Aussage` mit einem Wert. Anschließend wird der Wert dieser Variablen mit `lastIndexOf()` nach dem letzten Vorkommen von `Mensch` innerhalb der Zeichenkette gesucht. In der Variablen `Suche` wird der Rückgabewert von `lastIndexOf()` gespei-

chert. Im Beispiel wird 19 zurückgegeben, da die Zeichenfolge Mensch in der durchsuchten Zeichenkette zum letzten Mal an 20. Stelle der Zeichenkette beginnt. Vergleiche auch: `indexOf()`.

### **link()**

Formatiert eine Zeichenkette als Verweis. Hat den gleichen Effekt wie die HTML-Formatierung `<a href=>...</a>`. Erwartet als Parameter einen URI für das Verweisziel. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Homepage = "zu meiner Homepage";
  document.write(Homepage.link("home.htm"))
//-->
</script>
<br>
<script type="text/javascript">
<!--
  var document.write(document.referrer.link(document.referrer));
//-->
</script></body></html>
```

Das Beispiel im ersten Script belegt zunächst eine Variable `Homepage` mit einem Wert. Daran anschließend wird mit `document.write()` ein Verweis geschrieben. Dazu wird die Methode `link()` auf die Variable `Homepage` angewendet. Der Wert von `Homepage` wird dabei der Verweistext. Als Parameter wird das gewünschte Verweisziel übergeben, im Beispiel die Datei `home.htm`.

Das Beispiel im zweiten Script erzeugt dynamisch einen Verweis zu der Adresse, von wo aus der Anwender die aktuelle Seite aufgerufen hat. Dabei liefert die Eigenschaft `document.referrer` die Adresse der Herkunftsseite. Das Beispiel ist so konstruiert, dass hinterher die Adresse von `document.referrer` sowohl das Verweisziel als auch der Verweistext ist.

### **match()**

Durchsucht eine Zeichenkette mit Hilfe eines regulären Ausdrucks. Liefert Zeichenfolgen zurück, auf die der reguläre Ausdruck passt. Erwartet als Parameter den regulären Ausdruck. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Aussage = "Der Mensch ist des Menschen sein Feind";
  var Ergebnis = Aussage.match(/ Mensch.[+^\S] /);
  alert(Ergebnis);
//-->
</script>
<script type="text/javascript">
<!--
  Aussage = "Der Mensch ist des Menschen sein Feind";
  Ergebnis = Aussage.match(/ \bMensch\b|\bMenschen\b /g);
```

```

if(Ergebnis)
  for(i=0;i<Ergebnis.length;++i)
    alert(Ergebnis[i]);
//-->
</script>
<script type="text/javascript">
<!--
  Aussage = "Wir wollen weisse Waesche waschen";
  Ergebnis = Aussage.match(/w/gi);
  if(Ergebnis)
    alert(Ergebnis.length + " gefunden");
//-->
</script>
</body></html>

```

Das erste Script im Beispiel belegt eine Variable `Aussage` mit einem Wert. Auf den gespeicherten Wert dieser Variablen wird mit `match()` ein regulärer Ausdruck angewendet. Dieser findet die Zeichenfolge »Mensch«, wenn noch zwei weitere Zeichen folgen, aber nicht, wenn ein Leerzeichen dabei ist. Auf diese Weise wird nur das hintere Wort »Menschen« gefunden. Das Ergebnis wird in einem Meldungsfenster ausgegeben.

Das zweite Script im Beispiel wendet auf den gespeicherten Wert der Variablen `Aussage` mit `match()` einen anderen regulären Ausdruck an. Dieser findet die Zeichenfolge »Mensch«, aber auch die Zeichenfolge »Mensch«. Hinter dem Ausdruck ist außerdem mit `g` markiert, dass global gesucht werden soll. Dadurch werden alle Fundstellen zurückgegeben. Da es sich um mehrere Fundstellen handeln kann, ist der zurückgegebene Wert kein einfacher Wert, sondern ein Array aus Werten. Die Variable `Ergebnis`, in der der Rückgabewert im Beispiel gespeichert wird, muss deshalb anschließend als Array behandelt werden. Das Array-Objekt wird jedoch nur erzeugt, wenn es Suchtreffer gibt. Deshalb ist es sicherer, folgende Anweisungen von der Abfrage abhängig zu machen, ob das Objekt existiert. Mit `if(Ergebnis)` ist eine solche Abfrage möglich. Im Beispiel werden dann in einer `for`-Schleife alle Suchtreffer ausgegeben. Dieses Script wird von Opera 5.12 nicht interpretiert.

Das dritte Script im Beispiel belegt die Variable `Aussage` mit einem neuen Wert, in dem der Buchstabe `w` oft vorkommt, mal klein- und mal großgeschrieben. Auf den gespeicherten Wert dieser Variablen wird mit `match()` ein regulärer Ausdruck angewendet. Dieser sucht nach dem Vorkommen von `w`, und zwar wieder, wie im zweiten Script, global. Es werden also alle Fundstellen als Array zurückgegeben. Im Beispiel sollen jedoch sowohl klein- als auch großgeschriebene `w` gefunden werden. Deshalb wird hinter dem regulären Ausdruck noch zusätzlich ein kleines `i` notiert. Dies bewirkt, dass Groß-/Kleinschreibung nicht beachtet wird. Zur Kontrolle wird im Beispiel die Anzahl der Fundstellen ausgegeben.

Näheres über reguläre Ausdrücke finden Sie beim `RegExp`-Objekt (Kapitel 23.30). Näheres über Arrays wird beim `Array`-Objekt beschrieben (Kapitel 23.21). Um nicht gefundene Zeichenfolgen, sondern nur Positionen gefundener Ausdrücke zu ermitteln, können Sie die Methode `search()` verwenden.



Durchsucht eine Zeichenkette mit Hilfe eines regulären Ausdrucks. Ersetzt Zeichenfolgen, auf die der reguläre Ausdruck passt. Erwartet folgende Parameter:

- *reg. Ausdruck* = Ein regulärer Ausdruck, der Suchtreffer liefern kann.
- *Ersetzen durch* = Eine Zeichenkette, durch die gefundene Stellen ersetzt werden sollen.

Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var vormalis = "Elke Hinz";
  var jetzt = vormalis.replace(/Hinz/, "Kunz, geb. Hinz");
  document.write(jetzt);
//-->
</script></body></html>
```

Das Script im Beispiel belegt eine Variable `vormalis` mit einem Namen. Auf den gespeicherten Namen wird mit `replace()` ein Suche-Ersetze-Vorgang angewendet. Der erste Parameter, der `replace()` übergeben wird, ist ein regulärer Ausdruck. Dieser findet die Zeichenfolge »Hinz«. Ein Suchtreffer bewirkt das Ersetzen dieser Zeichenfolge durch »Kunz, geb. Hinz«. Das Ergebnis wird zur Kontrolle in die Datei geschrieben. Näheres über reguläre Ausdrücke und weitere Beispiele für diese Methode finden Sie beim `RegExp`-Objekt (Kapitel 20.30).



Durchsucht eine Zeichenkette mit Hilfe eines regulären Ausdrucks. Liefert -1 zurück, wenn der reguläre Ausdruck nicht passt. Wenn er passt, wird die Position des ersten Vorkommens zurückgeliefert. Erwartet als Parameter den regulären Ausdruck. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var Aussage = "Wir wollen weisse Waesche waschen";
  var Ergebnis = Aussage.search(/weiss.+/);
  if(Ergebnis != -1)
    alert("gefunden");
//-->
</script>
</body></html>
```

Das Beispiel belegt die Variable `Aussage` mit einem Wert. Auf den gespeicherten Wert dieser Variablen wird mit `search()` ein regulärer Ausdruck angewendet. Dieser sucht nach dem Vorkommen von »weiss«, gefolgt von einem beliebigen Inhalt. Der Rückgabewert von `search()` wird in der Variablen `Ergebnis` gespeichert. Wenn diese am Ende ungleich -1 ist, gab es zu dem regulären Ausdruck Suchtreffer. Im Beispiel wird in diesem Fall einfach eine entsprechende Meldung ausgegeben. Näheres über reguläre Ausdrücke finden Sie beim `RegExp`-Objekt (Kapitel 20.30). Um nicht Positionen gefundener Ausdrücke als Rückgabewerte zu erhalten, sondern die Ausdrücke selbst, können Sie die Methode `match()` verwenden.




***slice()***

Extrahiert aus einer Zeichenkette eine Teilzeichenkette. Gibt die extrahierte Zeichenkette zurück. Erwartet folgende Parameter:

- *Index Anfang* = Position des ersten zu extrahierenden Zeichens in der Zeichenkette, wobei bei 0 zu zählen begonnen wird.
- *Index Ende* = Position des ersten nicht mehr zu extrahierenden Zeichens in der Zeichenkette, wobei bei 0 zu zählen begonnen wird.

Sie können den zweiten Parameter auch weglassen. Dann wird bis zum Ende der Zeichenkette extrahiert. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Begriff = "Garnele";
  var Extrakt = Begriff.slice(0,4);
  alert(Extrakt);
//-->
</script></body></html>
```

Das Beispiel belegt die Variable `Aussage` mit einem Wert. Aus dem Wert dieser Variablen werden mit `slice()` die ersten vier Zeichen, also die Zeichen mit den Positionswerten 0, 1, 2 und 3, extrahiert. Der Rückgabewert der Funktion, die extrahierte Zeichenkette, wird im Beispiel in der Variablen `Extrakt` gespeichert. Diese wird zur Kontrolle als Meldung ausgegeben.




***small()***

Formatiert eine Zeichenkette als kleingeschriebenen Text, genau wie die HTML-Formatierung `<small>...</small>`. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var update = document.lastModified;
  document.write(update.small());
//-->
</script></body></html>
```

Im Beispiel ermittelt ein JavaScript mit Hilfe von `document.lastModified`, wann die Datei zuletzt geändert wurde. In der Variablen `update` steht dann eine entsprechende Zeichenkette. Mit Hilfe von `document.write()` wird der Wert von `update` in die Datei geschrieben. Die Formatierung dabei entspricht der HTML-Formatierung `<small>...</small>`.




***split()***

Zerlegt Zeichenketten in mehrere Teilzeichenketten. Die erzeugten Teilzeichenketten werden in einem Array gespeichert. Erwartet als Parameter ein Begrenzerzeichen oder eine Begrenzerzeichenfolge, die als Trennmerkmal für die Teilzeichenketten verwendet werden soll. Optional kann

als zweiter Parameter noch übergeben werden, wie viele Split-Vorgänge maximal erzeugt werden sollen. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Satz = "Wer nicht vom rechten Weg abkommt bleibt auf der Strecke";
  var Woerter = Satz.split(" ");
  document.write("Ein Satz mit " + Woerter.length + " W&ouml;rtern.<br>");
  document.write("Das 5. Wort ist " + Woerter[4]);
  //-->
</script></body></html>
```

Das Beispiel belegt die Variable `Satz` mit einem Wert. Aus dem Wert dieser Variablen werden mit `split()` die einzelnen Wörter extrahiert. Dazu wird der Methode als Parameter ein Leerzeichen übergeben. Bei jedem Leerzeichen im Satz wird dadurch eine neue Teilzeichenkette erzeugt.

Der Rückgabewert der Funktion, die extrahierte Zeichenkette, wird im Beispiel in der Variablen `Woerter` gespeichert. Dies ist keine gewöhnliche Variable, sondern ein Array. Zu Testzwecken schreibt das Beispiel in die Datei, wie viele Elemente der Array hat (`Woerter.length`) und welches das 5. Wort ist. Dies ist das Wort mit dem Array-Index 4, da auch in diesem Fall bei 0 zu zählen begonnen wird. Näheres über Arrays finden Sie beim `Array`-Objekt (Kapitel 23.30).

### **strike()**

Formatiert eine Zeichenkette als durchgestrichenen Text, genau wie die HTML-Formatierung `<strike>...</strike>`. Im Beispiel schreibt ein JavaScript mit `document.write()` den URI der aktuellen Datei. Die Formatierung dabei entspricht der HTML-Formatierung `<strike>...</strike>`.

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  document.write(window.location.href.strike());
  //-->
</script></body></html>
```

### **sub()**

Formatiert eine Zeichenkette als tiefgestellten Text, genau wie die HTML-Formatierung `<sub>...</sub>`. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Tief0 = "0";
  document.write("ISDN und S" + Tief0.sub() + "-Bus");
  //-->
</script></body></html>
```

Im Beispiel wird eine Variable `Tief0` definiert, die das Zeichen 0 speichert. Mit Hilfe von `document.write()` wird dann ein Text in die Datei geschrieben. Die Formatierung der 0 entspricht dabei der HTML-Formatierung `<sub>...</sub>`.

### **substr()**

Extrahiert aus einer Zeichenkette eine Teilzeichenkette ab einer bestimmten Zeichenposition und mit einer bestimmten Länge. Gibt die extrahierte Zeichenkette zurück. Erwartet folgende Parameter:

- *Index Anfang* = Position des ersten zu extrahierenden Zeichens in der Zeichenkette, wobei bei 0 zu zählen begonnen wird.
- *Anzahl Zeichen* = Anzahl Zeichen, die extrahiert werden sollen.

Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Begriff = "Donaudampfschiffahrt";
  var Teil = Begriff.substr(5,11);
  alert(Teil);
  //-->
</script></body></html>
```

Das Beispiel belegt die Variable `Begriff` mit einem Wert. Aus dem Wert dieser Variablen wird mit `substr()` ein Teil extrahiert, nämlich ab Zeichen 6 (Index 5), und dann die folgenden 11 Zeichen. Der Rückgabewert der Funktion, die extrahierte Zeichenkette, wird im Beispiel in der Variablen `Teil` gespeichert. Zur Kontrolle wird diese Variable als Meldungsfenster ausgegeben. Sie enthält den Wert `dampfschiff`.

### **substring()**

Extrahiert aus einer Zeichenkette eine Teilzeichenkette ab einer bestimmten Zeichenposition und bis zu einer bestimmten Zeichenposition. Gibt die extrahierte Zeichenkette zurück. Erwartet folgende Parameter:

- *Index Anfang* = Position des ersten zu extrahierenden Zeichens in der Zeichenkette, wobei bei 0 zu zählen begonnen wird.
- *Index Ende* = Position des ersten nicht mehr zu extrahierenden Zeichens in der Zeichenkette, wobei bei 0 zu zählen begonnen wird.

Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Begriff = "Donaudampfschiffahrt";
  var Teil = Begriff.substring(5,16);
  alert(Teil);
  //-->
</script></body></html>
```

Das Beispiel belegt die Variable `Begriff` mit einem Wert. Aus dem Wert dieser Variablen wird mit `substring()` ein Teil extrahiert, nämlich ab Zeichen 6 (Index 5), und bis vor Zeichen 17 (Index 16). Der Rückgabewert der Funktion, die extrahierte Zeichenkette, wird im Beispiel in der Variablen `Teil` gespeichert. Zur Kontrolle wird diese Variable als Meldungsfenster ausgegeben. Sie enthält den Wert `dampfschiff`.



Formatiert eine Zeichenkette als hochgestellten Text, genau wie die HTML-Formatierung `<sup>...</sup>`. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<script type="text/javascript">
<!--
  var Hoch2 = "2";
  document.write("10" + Hoch2.sup() + " = " + Math.pow(10,2));
//-->
</script></body></html>
```

Im Beispiel wird eine Variable `Hoch2` definiert, die das Zeichen 2 speichert. Mit Hilfe von `document.write()` wird dann ein Text in die Datei geschrieben. Die Formatierung der 2 entspricht dabei der HTML-Formatierung `<sup>...</sup>`.



Wandelt alle Großbuchstaben, die in einer Zeichenkette enthalten sind, in Kleinbuchstaben um. Erwartet die Zeichenkette, die so behandelt werden soll, als Parameter. Gibt die neue Zeichenkette zurück. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function MachKlein() {
  var neu = document.Test.Eingabe.value.toLowerCase();
  document.Test.Eingabe.value = neu;
}
//-->
</script>
</head><body>
<form name="Test" action="">
<input name="Eingabe">
<input type="button" value="klein machen" onClick="MachKlein()">
</form>
</body></html>
```

Im Beispiel wird ein Formular mit einem Eingabefeld und einem Button definiert. Wenn der Anwender auf den Button klickt, wird die Funktion `MachKlein()` aufgerufen. Diese Funktion wendet die Methode `toLowerCase()` auf das Eingabefeld des Formulars an. Das Ergebnis wird in der Variablen `neu` gespeichert. Anschließend wird dem Formulareingabefeld der Wert von `neu` zugewiesen. Dadurch werden alle eventuell eingegebenen Großbuchstaben in dem Eingabefeld zu Kleinbuchstaben.



 **toUpperCase()**

Wandelt alle Kleinbuchstaben, die in einer Zeichenkette enthalten sind, in Großbuchstaben um. Erwartet die Zeichenkette, die so behandelt werden soll, als Parameter. Gibt die neue Zeichenkette zurück. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function MachGross() {
    neu = document.Test.Eingabe.value.toUpperCase();
    document.Test.Eingabe.value = neu;
}
//-->
</script>
</head><body>
<form name="Test" action="">
<input name="Eingabe">
<input type="button" value="gross machen" onClick="MachGross()">
</form>
</body></html>
```

Im Beispiel wird ein Formular mit einem Eingabefeld und einem Button definiert. Wenn der Anwender auf den Button klickt, wird die Funktion `MachGross()` aufgerufen. Diese Funktion wendet die Methode `toUpperCase()` auf das Eingabefeld des Formulars an. Das Ergebnis wird in der Variablen `neu` gespeichert. Anschließend wird dem Formulareingabefeld der Wert von `neu` zugewiesen. Dadurch werden alle eventuell eingegebenen Kleinbuchstaben in dem Eingabefeld zu Großbuchstaben.

## 20.33 Objektunabhängige Funktionen

|                                   |                                       |
|-----------------------------------|---------------------------------------|
| <code>decodeURI()</code>          | (kodierte(n) URI dekodieren)          |
| <code>decodeURIComponent()</code> | (kodierte(n) URI dekodieren – II)     |
| <code>encodeURI()</code>          | (URI kodieren)                        |
| <code>encodeURIComponent()</code> | (URI kodieren – II)                   |
| <code>eval()</code>               | (Ausdruck interpretieren)             |
| <code>escape()</code>             | (ASCII-Zeichen in Zahlen umwandeln)   |
| <code>isFinite()</code>           | (auf numerischen Wertebereich prüfen) |
| <code>isNaN()</code>              | (auf numerischen Wert prüfen)         |
| <code>parseFloat()</code>         | (in Kommazahl umwandeln)              |
| <code>parseInt()</code>           | (in Ganzzahl umwandeln)               |
| <code>Number()</code>             | (auf numerischen Wert prüfen)         |
| <code>String()</code>             | (in Zeichenkette umwandeln)           |
| <code>unescape()</code>           | (Zahlen in ASCII-Zeichen umwandeln)   |

### Allgemeines zu objektunabhängigen Funktionen

Objektunabhängige Funktionen sind im Gegensatz zu selbst definierten Funktionen in JavaScript bereits vordefiniert. Das heißt, Sie können diese Funktionen jederzeit aufrufen. Es handelt sich um bestimmte, zum Teil mächtige JavaScript-Befehle, die in keines der JavaScript-Objekte passen und deshalb nicht der objektorientierten Richtung von JavaScript folgen.



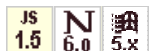
**decodeURI()**

Dekodiert einen URI, der mit `encodeURIComponent()` kodiert wurde. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
Adresse = "http://www.my.de/cgi-bin/script.pl?Text=der übergebene Text&Name=Stefan
Münz";
document.write("<b>Adresse unkodiert:</b> " + Adresse + "<br>");
Adresse = encodeURIComponent(Adresse);
document.write("<b>Adresse kodiert:</b> " + Adresse + "<br>");
Adresse = decodeURI(Adresse);
document.write("<b>Adresse wieder dekodiert:</b> " + Adresse + "<br>");
//-->
</script>
</body></html>
```

Das Beispiel definiert eine typische Adresse mit Übergabeparametern an ein Script. Zunächst schreibt es die Adresse wie sie ist ins Dokument. Dann wird die Adresse mit `encodeURIComponent()` behandelt und zur Kontrolle ausgegeben. Schließlich wird nach dem gleichen Schema `decodeURI()` angewendet und die Adresse nochmals ausgegeben.

**Beachten Sie:** Opera 5.12 interpretiert dieses Methode nicht.



**decodeURIComponent()**

Wie `decodeURI()`. Sollte aber nur auf Adressen oder Adressteile angewendet werden, die mit `encodeURIComponent()` kodiert wurden.



**encodeURIComponent()**

Verschlüsselt einen URI so, dass alle Sonderzeichen in ASCII-Zeichensequenzen umgewandelt werden. Besonders wichtig ist dies für Aufrufe von CGI-Scripts mit Parametern. Denn HTML 4.0 verlangt beispielsweise, im `href`-Attribut von Verweisen enkodierte Zeichenketten zu verwenden, sofern die Adressen Sonderzeichen enthalten.

Kodiert alle außer den folgenden Zeichen:

A bis Z

a bis z

- \_ . ! ~ \* ' ( )

, / ? : @ & = + \$

Kodiert werden also beispielsweise deutsche Umlaute und Sonderzeichen, auch Leerzeichen, eckige und geschweifte Klammern usw. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Adresse = encodeURI("http://www.my.de/cgi-bin/script.pl?Text=der übergebene
Text&Name=Stefan Münz");
document.write("<a href=\"" + Adresse + "\"">Verweis</a>");
//-->
</script>
</body></html>
```

Das Beispiel ruft die Funktion `encodeURI()` auf und übergibt ihr einen typischen URI mit Parametern. Der Rückgabewert der Funktion wird in der Variablen `Adresse` gespeichert. Anschließend wird dynamisch ein Verweis in die Datei geschrieben, wobei die zuvor enkodierte Adresse ins `href`-Attribut eingefügt wird.

**Beachten Sie:** Opera 5.12 interpretiert dieses Methode nicht.

#### **`encodeURIComponent()`**

Wie `encodeURI()`, aber mit dem Unterschied, dass auch folgende Zeichen kodiert werden: `, / ? : @ & = + $`

#### **`escape()`**

Wandelt Steuersequenzen (Steuerzeichen mit den ASCII-Werten 0 bis 31) und Sonderzeichen wie z.B. deutsche Umlaute in ihre ASCII-Zahlenwerte um, und zwar in hexadezimaler Form. Setzt vor jeden Wert das Trennzeichen `»%«` und gibt die so erzeugte Zeichenkette zurück. Bei anderen Zeichen als Steuer- und Sonderzeichen gibt `escape()` die Zeichen unverändert zurück. Es handelt sich also um eine Funktion, die Steuer- und Sonderzeichen aus Werten entfernt und in Zeichenfolgen übersetzt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Beispiel = "\n\r\t";
document.write(escape(Beispiel));
//-->
</script></body></html>
```

Das Beispiel definiert eine Variable `Beispiel`, der verschiedene Steuerzeichen zugewiesen werden (siehe dazu den Abschnitt 18.5 über Steuerzeichen). Diese Variable wird der Funktion `escape()` als Parameter übergeben. Das Ergebnis wird zur Kontrolle ins Dokumentfenster geschrieben.

**Beachten Sie:** Diese Funktion wird in Zukunft durch `encodeURI()` ersetzt!



Interpretiert ein zu übergebendes Argument und gibt das Ergebnis zurück. Wenn das übergebene Argument als Rechenoperation interpretierbar ist, wird die Operation berechnet und das Ergebnis zurückgegeben. Dabei sind auch komplexe Rechenausdrücke mit Klammerung möglich. Diese Funktionalität ist sehr praktisch, um als Zeichenketten notierte Rechenausdrücke mit einem einzigen Befehl errechnen zu lassen.

Wenn das übergebene Argument als Objekt oder Objekteigenschaft interpretiert werden kann, wird das Objekt bzw. die Objekteigenschaft zurückgegeben. Dies ist wichtig, wenn eine Zeichenkette (z.B. ein Formularfeldinhalt oder ein einer Funktion übergebener Parameter) als Objekt interpretiert werden soll. Erzeugt eine Fehlermeldung, wenn der übergebene Ausdruck nicht interpretierbar ist. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<form action="">
<input size="30" name="Eingabe">
<input type="button" value="Berechnen"
onClick="alert(eval(this.form.Eingabe.value))">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einem Eingabefeld und einem Button. In dem Eingabefeld kann der Anwender eine Rechenoperation eingeben, zum Beispiel  $1+2+3+4+5+6+7+8+9$  oder  $12.78*5.13$ . Beim Klick auf den Button wird der Wert aus dem Formular mit `eval()` berechnet und in einem Meldungsfenster ausgegeben.



Ermittelt, ob ein Wert sich innerhalb des Zahlenbereichs befindet, den JavaScript verarbeiten kann, also aus Sicht von JavaScript eine gültige Zahl darstellt. Gibt `true` zurück, wenn der Wert eine gültige Zahl ist, und `false`, wenn es keine Zahl ist bzw. die Zahl sich außerhalb des Wertebereiches von JavaScript befindet. Sowohl ganze Zahlen als auch Kommazahlen werden als gültige Zahlen erkannt. Bei Kommazahlen muss das Dezimalzeichen ein Punkt sein. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var Zahl=Number.MAX_VALUE;
if (!isFinite(Zahl*2)) alert("Die Zahl ist nicht zu verarbeiten.");
//-->
</script>
</head><body>
</body></html>
```

Das Beispiel definiert eine Variable `Zahl`, der als Wert `Number.MAX_VALUE` die größte mögliche Zahl zugewiesen wird. In der anschließenden Abfrage wird geprüft, ob das Produkt dieser Zahl mit 2 sich innerhalb des verarbeitbaren Zahlenbereiches befindet. Ist dies nicht der Fall, so wird eine entsprechende Warnmeldung ausgegeben.




**isNaN()**

Ermittelt, ob ein zu übergebender Wert eine ungültige Zahl ist (NaN = Not a Number). Gibt `true` zurück, wenn der Wert **keine** Zahl ist, und `false`, wenn es eine Zahl ist. Sowohl ganze Zahlen als auch Kommazahlen werden als gültige Zahlen anerkannt. Bei Kommazahlen muss das Dezimalzeichen ein Punkt sein. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function CheckZahl(Wert)
{
  if(isNaN(Wert) == true)
  {
    alert(Wert + " ist keine Zahl!");
    return false;
  }
  else return true;
}
//-->
</script>
</head><body>
<form action="">
<input size="30" name="Eingabe">
<input type="button" value="Check"
onClick="CheckZahl(this.form.Eingabe.value)">
</form>
</body></html>
```

Das Beispiel enthält ein Formular mit einem Eingabefeld und einem Button. Beim Klick auf den Button wird die Funktion `CheckZahl()` aufgerufen. Der Funktion wird der Wert aus dem Eingabefeld des Formulars als Parameter übergeben. `CheckZahl()` prüft, ob der übergebene Wert eine Zahl ist. Dazu wird `isNaN()` auf den übergebenen Wert angewendet. Wenn es keine Zahl ist, wird eine Fehlermeldung ausgegeben.

**Beachten Sie:** Für diese Methode sind auch Zahlen, die außerhalb des Zahlenbereichs von JavaScript liegen, gültige Zahlen. Zur Überprüfung der Größe von Zahlen müssen Sie die Methode `isFinite()` verwenden.




**parseFloat()**

Wandelt eine zu übergebende Zeichenkette in eine Zahl um und gibt diese als numerischen Wert zurück. Wenn sich die Zahl als Kommazahl interpretieren lässt, wird dies berücksichtigt. Als Dezimalzeichen wird jedoch nur der Punkt interpretiert. Sinnvoll, um z.B. Anwendereingaben in Zahlen umzuwandeln, mit denen man anschließend rechnen kann.

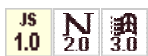
Gibt `NaN` (Not a Number) zurück, wenn die Zeichenkette mit Zeichen beginnt, die sich nicht als Teil einer Zahl interpretieren lassen. Wenn die Zeichenkette weiter hinten ungültige Zeichen enthält, wird die Zahl bis zum ersten ungültigen Zeichen interpretiert und der interpretierte Teil zurückgegeben. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
var Elemente = new Array("18","18.1","18.9","abc","1a");
document.write("<h1>Anwendung von <i>parseFloat()</i></h1>");
for(var i=0;i<Elemente.length;++i)
  document.write(Elemente[i] + " = <b> " + parseFloat(Elemente[i]) + "</b><br>");
//-->
</script></body></html>

```

Das Beispiel definiert einen Array mit verschiedenen Elementen, die teils ganze, teils Kommazahlen, teils gar keine Zahlen darstellen. Dann werden in einer `for`-Schleife alle Elemente des Arrays mit der `write()`-Methode dynamisch in die Datei geschrieben. Dabei wird jedoch die Funktion `parseFloat()` auf das jeweilige Array-Element angewendet.



### ***parseInt()***

Wandelt eine zu übergebende Zeichenkette in eine Ganzzahl um und gibt diese als Ergebnis zurück. Sinnvoll, um z.B. Anwendereingaben in Zahlen umzuwandeln, mit denen man anschließend rechnen kann. Gibt `NaN` (Not a Number) zurück, wenn die Zeichenkette mit Zeichen beginnt, die sich nicht als Teil einer Zahl interpretieren lassen. Wenn die Zeichenkette weiter hinten ungültige Zeichen enthält, wird die Zahl bis zum ersten ungültigen Zeichen interpretiert und der interpretierte Teil zurückgegeben. Das gilt bei `parseInt()` auch für den Punkt. Ein Beispiel:

```

<html><head><title>Test</title>
</head><body>
<script language="JavaScript" type="text/javascript">
<!--
var Elemente = new Array("18","18.1","18.9","abc","1a");
document.write("<h1>Anwendung von <i>parseInt()</i></h1>");
for(var i=0;i<Elemente.length;++i)
  document.write(Elemente[i] + " = <b> " + parseInt(Elemente[i]) + "</b><br>");
//-->
</script></body></html>

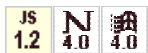
```

Das Beispiel definiert einen Array mit verschiedenen Elementen, die teils ganze, teils Kommazahlen, teils gar keine Zahlen darstellen. Dann werden in einer `for`-Schleife alle Elemente des Arrays mit der `write()`-Methode dynamisch in die Datei geschrieben. Dabei wird jedoch die Funktion `parseInt()` auf das jeweilige Array-Element angewendet.

**Beachten Sie:** Die Methode `parseInt()` erlaubt einen zweiten, optionalen Parameter *radix*. Dieser Parameter enthält die Basis des verwendeten Zahlensystems. Der Aufruf `parseInt(string, radix)` ermittelt aus dem String, der eine Zahl eines anderen Zahlensystems darstellt, die dazugehörige Dezimalzahl. Die Methode `parseInt()` kann damit gleichzeitig als Umrechner von Zahlen anderer Zahlensysteme in das dezimale Zahlensystem verwendet werden. So ergibt z.B. der Aufruf `parseInt("11111", 2)` die Zahl 31. Der optionale Parameter kann Werte zwischen 2 und 36 annehmen. Mit der Methode `toString()` des `Number`-Objektes können Sie umgekehrt Dezimalzahlen in Zeichenketten von Zahlen anderer Zahlensysteme konvertieren.

Dabei ist – im Gegensatz zur Methode `parseInt()` – auch das Umwandeln von Kommazahlen möglich.

Besonders im Bereich der Datumsberechnung werden häufig Zahlen mit führender Null benutzt. Einfache Aufgaben wie `alert(08*60)` liefern dann scheinbar falsche Ergebnisse oder gar Fehlermeldungen. Des Rätsels Lösung dafür ist, dass JavaScript Zahlen mit führender 0 als Oktalzahlen interpretiert, nicht als Dezimalzahlen. Mit einer Anweisung wie `alert(parseInt("08",10)*60)` können Sie das Problem vermeiden und erzwingen, dass der Wert dezimal interpretiert wird.



### **Number()**

Konvertiert den Inhalt eines Objekts in eine Zahl und gibt die Zahl zurück. Erwartet den Objektinhalt als Parameter. Wenn sich der übergebene Parameter nicht als Zahl interpretieren lässt, wird NaN (Not a Number) zurückgegeben. Vor allem brauchbar in Verbindung mit dem Date-Objekt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var d = new Date ("March 15, 1998 08:27:00");
  document.write(Number(d));
//-->
</script></body></html>
```

Das Beispiel definiert ein neues Datumobjekt und initialisiert es mit einem Wert im GMT-Format. Das Datumobjekt wird in der Variablen `d` gespeichert. Anschließend wird diese Variable der Funktion `Number()` als Parameter übergeben. Die Funktion `Number()` hat dabei die gleiche Wirkung wie die Objektmethode `d.getTime()`, d.h. es wird die Anzahl Millisekunden zwischen dem 1.1.1970 und dem in `d` gespeicherten Zeitpunkt errechnet. Im Beispiel wird das Ergebnis zur Kontrolle in die Datei geschrieben.



### **String()**

Konvertiert den Inhalt eines Objekts in eine Zeichenkette und gibt die Zeichenkette zurück. Vor allem brauchbar in Verbindung mit dem Date-Objekt. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script type="text/javascript">
<!--
  var d = new Date ();
  document.write(String(d));
//-->
</script></body></html>
```

Das Beispiel definiert ein neues Datumobjekt mit dem aktuellen Zeitpunkt. Das Datumobjekt wird in der Variablen `d` gespeichert. Anschließend wird diese Variable der Funktion `String()` als Parameter übergeben. Die Funktion erzeugt daraus eine Zeichenkette, die den gespeicherten

Zeitpunkt im GMT-Format darstellt. Im Beispiel wird das Ergebnis zur Kontrolle in die Datei geschrieben.

### **unescape()**

Wandelt alle Zeichen der zu übergebenden Zeichenkette in normale ASCII-Zeichen um und gibt die so erzeugte Zeichenkette zurück. Die zu übergebende Zeichenkette muss für jedes umzuwandelnde Zeichen ein Prozentzeichen (%) und den Hexadezimalwert des Zeichens in der ASCII-Zeichentabelle enthalten. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<script language="JavaScript" type="text/javascript">
<!--
var Beispiel = "%53%74%65%66%61%6E";
document.write(unescape(Beispiel));
//-->
</script></body></html>
```

Das Beispiel definiert eine Variable namens `Beispiel` und weist ihr eine Zeichenkette zu. Die Zeichenkette besteht aus einer Reihenfolge von hexadezimal kodierten Zeichen. Jedes Zeichen beginnt mit einem Prozentzeichen, dahinter folgt der Hexadezimalwert des Zeichens. Diese Variable wird der Funktion `unescape()` als Parameter übergeben. Das Ergebnis wird zur Kontrolle in die Datei geschrieben. Im Beispiel wird das Wort »Stefan« erzeugt.

**Beachten Sie:** Diese Funktion wird in Zukunft durch `decodeURI()` ersetzt!



## 21 Dynamisches HTML

### 21.1 Allgemeines zu Dynamischem HTML

#### 21.1.1 Das »Dynamische« am Dynamischen HTML

Dynamisches HTML (engl. »Dynamic HTML« oder abgekürzt »DHTML«) ist eine Erfindung von Marktstrategen, sagen Kritiker. In der Tat ist Dynamisches HTML keine klassische HTML-Erweiterung in Gestalt neuer HTML-Elemente. Es ist auch keine neue Sprache. Dynamisches HTML ist vielmehr der Sammelbegriff für verschiedene Lösungen, um dem Autor einer Web-Seite zu ermöglichen, Elemente der Web-Seite während der Anzeige dynamisch zu ändern, sei es automatisch oder durch Einwirken des Anwenders. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p id="Absatz"
onClick="document.getElementById('Absatz').firstChild.nodeValue='Sehen Sie!'">
Ich bin dynamisch</p>
</body></html>
```

Das kleine Beispiel zeigt Dynamisches HTML nach dem Document Object Model (DOM) in seiner JavaScript-Umsetzung, das in dieser Form allerdings nur mit neueren Browsern (Internet Explorer ab V5.x und Netscape ab V6.x) funktioniert. Es dient hier nur dem Zweck, das »Typische« an Dynamischem HTML zu verdeutlichen. In dem Beispiel wird mit `<p>...</p>` ein gewöhnlicher HTML-Textabsatz definiert. Am Bildschirm steht dann: *Ich bin dynamisch*. Damit der Textabsatz sein Versprechen auch halten kann, enthält er im einleitenden `<p>`-Tag einen so genannten Event-Handler, im Beispiel den Event-Handler `onClick=`. Dieser Event-Handler tritt in Aktion, wenn der Anwender mit der Maus auf den angezeigten Text klickt. Im Beispiel verwandelt sich nach dem Anklicken des Textes dessen Inhalt in: *Sehen Sie!*.

Wie Sie an dem Beispiel erkennen, können Sie also bestimmte angezeigte Inhalte austauschen, ohne dafür jedes Mal vollständige neue Seiten laden zu müssen. Das ist eine charakteristische Eigenschaft von Dynamischem HTML. Eine Web-Seite verhält sich also plötzlich wie eine Anwendung, die einmal in den Arbeitsspeicher geladen wird und dann z.B. durch Interaktion mit dem Anwender das Geschehen am Bildschirm bestimmt.

Dynamisches HTML bietet Gestalten anspruchsvoller Web-Seiten schier unerschöpfliche Möglichkeiten bei der Kontrolle über das Geschehen am Bildschirm. Es fängt mit Kleinigkeiten wie dem obigen Beispiel an. Es geht weiter mit animierten Grafiken, die über den angezeigten Inhalt gleiten oder hüpfen – manche Web-Seiten versuchen, ganze Aquarien abzubilden. Und es geht hin bis zu der Möglichkeit, mit Hilfe von Grafiken und anderen Elementen komplexe, interaktive Spiele, Lernsituationen oder auch selbst ablaufende Animationen zu programmieren. Der Phantasie sind kaum Grenzen gesetzt. Der Anwender kann beim Anzeigen einer dynamischen HTML-Datei auch mit viel mehr Reaktionen konfrontiert werden als beim Betrachten einer

normalen, statischen Web-Seite. Das hat Vor- und Nachteile. Es kommt immer darauf an, wie man die Features nutzt, ob zur bloßen Selbstdarstellung und Angeberei oder zum Nutzen des Anwenders. Außerdem sollte sich jeder Entwickler darüber im Klaren sein, dass dynamische Web-Seiten nicht unbedingt für Suchmaschinen-Robots und erst recht nicht für sehbehinderte Anwender geeignet sind.

»Programmieren« ist auf jeden Fall bei allem angesagt, was mit Dynamischem HTML zu tun hat: fortgeschrittene Programmierkenntnisse in JavaScript sind Voraussetzung. Im Web finden Sie zwar fertige Lösungen zum Abkupfern, aber wer sich ernsthaft mit der Materie beschäftigen will, um eigene kreative Lösungen zu entwickeln, kommt um die Programmierung nicht herum. Das soll jedoch keine Entmutigung sein. Es gibt durchaus nützliche Anwendungsmöglichkeiten von Dynamischem HTML, die ohne allzugroßen Programmieraufwand auskommen.

### 21.1.2 Das Problem der inkompatiblen Modelle

Dynamisches HTML wird von Netscape ab Version 4.0 und vom MS Internet Explorer ab Version 4.0 interpretiert. Leider ist das Thema »Dynamic HTML« bei diesen beiden Browser-Versionen jedoch zu einem Schauplatz im Browser-Krieg verkommen. Denn die Realisierungen sind in beiden Browsern grundverschieden. Vieles muss für das 4er-Level der beiden Browser getrennt kodiert werden (mit doppeltem Kenntnisaufwand), sofern man für beide Browser schreiben will. Vieles funktioniert überhaupt nur mit dem einen oder mit dem anderen Browser. Vor allem kann sich das »echte« dynamische HTML (beliebiger Zugriff auf alle HTML-Elemente einer Web-Seite), wie es der Internet Explorer ab Version 4.0 bereits beherrscht, auf dem 4er-Level der Browser noch nicht richtig durchsetzen, da dort stets auf das wesentlich primitivere, Layer-orientierte Modell von Netscape Rücksicht genommen werden muss.

Mittlerweile zeichnet sich jedoch ein einheitlicher Standard ab. Zu verdanken ist dieser Standard dem W3-Konsortium, das auch HTML und CSS normiert. Für den Script-Sprachen-Zugriff auf Elemente einer Web-Seite hat das W3-Konsortium das **Document Object Model (DOM)** entworfen. Die neueren Versionen der führenden Browser setzen das DOM – wenn auch noch mit deutlichen Schwächen im Detail – in JavaScript um, sodass es in absehbarer Zeit wohl einfacher werden wird, Dynamisches HTML zu programmieren.

Dieses Buch stellt in den nächsten Abschnitten zunächst alle drei Modelle (Microsoft 4er-Level, Netscape 4er-Level und DOM) getrennt vor. Denn nur wenn Sie die Konzepte klar unterscheiden können, sind Sie in der Lage, Dynamisches HTML so zu schreiben, dass es mit verschiedenen Browsern funktioniert. Im zweiten Schritt können Sie dann sehen, wie sich Dynamisches HTML browserübergreifend realisieren lässt (Kapitel 22, *DHTML-Praxis-Lösungen*).

## 21.2 Das Document Object Model (DOM)

### 21.2.1 Allgemeines zum DOM

Das Document Object Model (DOM) ist eine vom W3-Konsortium verabschiedete Norm, die zunächst einmal den Script-Sprachen-Zugriff auf beliebige Elemente eines Auszeichnungssprachen-Dokuments beschreibt. Das DOM ist also weder selber eine eigene Script-Sprache noch ist es auf HTML beschränkt. Es definiert lediglich Objekte, Eigenschaften und Methoden, die eine

Script-Sprache umsetzen sollte, wenn sie sich DOM-fähig bezeichnen will. Anwendbar sollen diese Objekte, Eigenschaften und Methoden auf alle Dokumente sein, die in einer XML-gerechten Auszeichnungssprache geschrieben sind. Das W3-Konsortium betont ausdrücklich, dass das DOM nicht einfach nur eine Norm für »Dynamic HTML« sein soll. Das DOM ist auch nicht auf die Client-Seite, also etwa den Web-Browser beschränkt. Ebenso gut lässt es sich in serverseitigen Scripts, z.B. in CGI-Scripts einsetzen, um Dokumente dynamisch zu erzeugen.

Wie auch immer – das DOM ist die willkommene Lösung auch für das, was hier unter dynamischem HTML verstanden wird, also das nachträgliche dynamische Verändern von Inhalten einer im Browser-Fenster angezeigten Web-Seite. 1998 erstmals als offizielle W3-Empfehlung herausgegeben, wurde es im MS Internet Explorer 5.0 und in dem von der Mozilla-Entwicklergemeinde völlig neu programmierten Netscape-Browser, den Netscape selber unter der Version 6.0 anbot, implementiert (JavaScript-Version 1.5). Die Implementierung ist in den heutigen Browsern noch nicht vollständig. Immer wieder wird man als Programmierer auf rätselhafte Verhaltensweisen im einen oder anderen Browser stoßen. Dennoch ist die Richtung damit vorgegeben, wie Dynamisches HTML in Zukunft programmiert werden wird.

Eine Auszeichnungssprache wie HTML oder auch jede andere, XML-basierte Sprache ist als hierarchische Baumstruktur abbildbar. Die einzelnen Bestandteile einer solchen Baumstruktur werden als **Knoten** bezeichnet. Das zentrale Objekt des DOM ist deshalb das Objekt *node* (*node* = *Knoten*). Es gibt verschiedene Knotentypen. Innerhalb eines gewöhnlichen HTML-Dokuments gibt es auf jeden Fall drei wichtige Knotentypen, die Sie unterscheiden müssen: **Elementknoten**, **Attributknoten** und **Textknoten**.

Betrachten Sie zum Verständnis das folgende einfache HTML-Konstrukt:

```
<h1 align="center">Hallo Welt</h1>
```

In diesem Konstrukt gibt es einen Elementknoten, nämlich den Elementknoten des `h1`-Elements. Ferner gibt es einen Attributknoten, nämlich den des `align`-Attributs. Und schließlich gibt es so genannte »Zeichendaten«, die an zwei Stellen vorkommen: nämlich einmal als Inhalt des `h1`-Elements und einmal bei der Wertzuweisung an das `align`-Attribut. Diese Zeichendaten stellen selbst Knoten dar, nämlich Textknoten.

Ein weiteres Beispielkonstrukt soll das Verständnis vertiefen:

```
<h1 align="center">Hallo <i>Welt</i></h1>
```

In diesem Beispiel ist die Kursiv-Auszeichnung um das Wort `Welt` hinzugekommen. Wichtig ist dabei nun zu verstehen, wie die Knotenhierarchie aussieht: Das `h1`-Element ist in diesem kleinen Strukturbaum-Ausschnitt der Ausgangsknoten. Dieser Knoten hat nach den Regeln des DOM zwei **Kindknoten** und einen **assoziierten Knoten**: Die Kindknoten sind zum einen der Textknoten mit dem Wort `Hallo` und dem Leerzeichen dahinter, sowie der Elementknoten des `i`-Elements. Das `align`-Attribut im einleitenden `<h1>`-Tag ist dagegen kein Kindknoten, sondern ein assoziierter Knoten. Der Attributknoten hat jedoch selbst wiederum einen Kindknoten, nämlich den zugewiesenen Wert (`center`). Auch der Elementknoten des `i`-Elements hat wieder einen Kindknoten, nämlich den Textknoten seines Zeicheninhalts, also das Wort `Welt`.

Die Baumstruktur einer komplexen Web-Seite kann, wie sich aus diesen einfachen Beispielen erschließt, sehr umfangreich und tief verschachtelt sein. In einer Script-Sprache muss es aber möglich sein, möglichst schnell und effizient auf einzelne Knoten zugreifen zu können. Im DOM

gibt es daher drei wichtige Methoden des `document`-Objekts, um zumindest auf jeden beliebigen Elementknoten direkt zugreifen zu können:

- `getElementById()` kann auf Elemente zugreifen, die ein dokumentweit eindeutiges `id`-Attribut enthalten, z.B. `<p id="derAbsatz">hier steht der Text</p>`.
- `getElementsByName()` kann auf Elemente zugreifen, die einen Namen besitzen (er muss nicht unbedingt eindeutig sein), z.B. `<input name="Eingabefeld" type="text" size="40">`.
- `getElementsByTagName()` kann auf alle Elemente zugreifen in der Form: »liefere mit das 27. td-Element im Dokument«.

Ausgehend von einem ermittelten Elementknoten lässt sich dann schnell auf dessen Attribute und Inhalt zugreifen. Auch dafür stellt das DOM Eigenschaften und Methoden bereit. Dabei gibt es jedoch bei HTML-basierten Web-Seiten zwei Varianten. Einmal können Sie die Eigenschaften und Methoden des `node`-Objekts dazu verwenden, um auf Kindknoten und assoziierte Knoten eines Elements zuzugreifen. Zum anderen können Sie aber auch die HTML-Elementobjekte (Kapitel 20.5) benutzen. Vor allem für den Zugriff auf einzelne Attribute eines Elements sind die HTML-Elementobjekte gut geeignet.

Dynamisches HTML kann aber auch darin bestehen, neue Elemente mit Inhalten und Formatierungen zur Anzeigezeit einer Web-Seite neu zu erzeugen. Für diesen Zweck hält das DOM ebenfalls Methoden bereit.

In der Version 1.0 des DOM wurde nur der Zugriff auf Knoten geregelt. Der Zugriff auf zugewiesene Style-Eigenschaften und von Event-Handling ist dort noch ausgespart. In der DOM-Version 2.0 vom November 2000 sind jedoch auch diese für dynamische Web-Seiten wichtigen Gebiete behandelt. Es ist deshalb auf jeden Fall empfehlenswert, bei neu zu erstellenden dynamischen Web-Seiten von den im DOM geregelten Techniken auszugehen. Die älteren Modelle von Microsoft (Kapitel 21.3) und Netscape (Kapitel 21.4) sollten derzeit noch aus Gründen der Rückwärtskompatibilität eingesetzt werden, auf die Dauer jedoch langsam aus der Welt verschwinden, damit endlich ein einheitlicher Sprachenstandard das Programmieren dynamischer Web-Seiten erleichtert.

## 21.2.2 Elementinhalte dynamisch ändern

Es gibt viele sinnvolle Anwendungsfälle zum dynamischen Ändern von zunächst angezeigten HTML-Elementen. Das folgende Beispiel zeigt einen davon. In dem Beispiel wird eine nummerierte Liste von Großstädten zunächst alphabetisch sortiert ausgegeben. Unterhalb der Liste kann der Anwender jedoch mit Hilfe zweier Formular-Buttons zwischen alphabetischer Sortierung oder geographischer Sortierung wechseln. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var geoArray = new Array(6,7,0,1,4,3,8,9,2,5,11,12,10);

function ABC() {
  var Staedte = new Array();
```

```

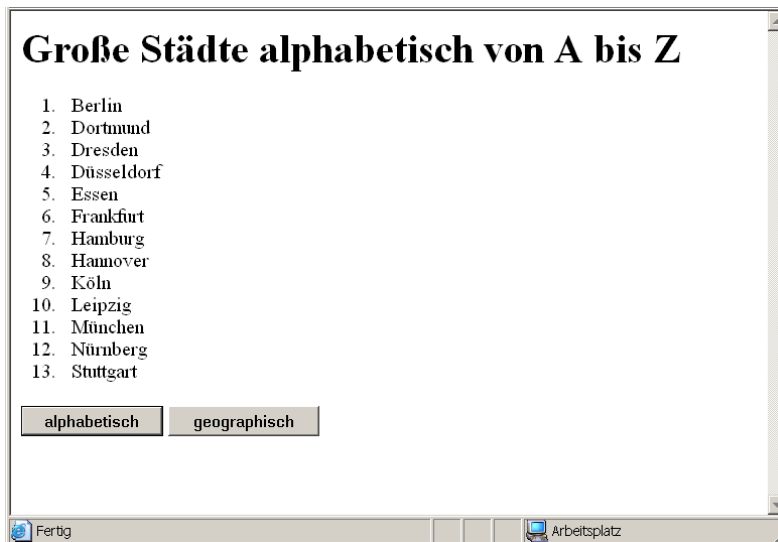
for(var i = 0; i < document.getElementsByTagName("li").length; i++)
    Staedte.push(document.getElementsByTagName("li")[i].firstChild.nodeValue);
Staedte.sort();
for(i = 0; i < document.getElementsByTagName("li").length; i++)
    document.getElementsByTagName("li")[i].firstChild.nodeValue = Staedte[i];
document.getElementById("Art").firstChild.nodeValue = "alphabetisch von A bis Z";
}

function GEO() {
    var Staedte = new Array();
    for(var i = 0; i < document.getElementsByTagName("li").length; i++)
        Staedte.push(document.getElementsByTagName("li")[i].firstChild.nodeValue);
    for(i = 0; i < document.getElementsByTagName("li").length; i++)
        document.getElementsByTagName("li")[i].firstChild.nodeValue =
Staedte[geoArray[i]];
    document.getElementById("Art").firstChild.nodeValue = "geographisch von Nord nach
Süd";
}
//-->
</script>
</head><body>
<h1>Große Städte <span id="Art">alphabetisch von A bis Z</span></h1>
<ol>
<li>Berlin</li>
<li>Dortmund</li>
<li>Dresden</li>
<li>Düsseldorf</li>
<li>Essen</li>
<li>Frankfurt</li>
<li>Hamburg</li>
<li>Hannover</li>
<li>Köln</li>
<li>Leipzig</li>
<li>München</li>
<li>Nürnberg</li>
<li>Stuttgart</li>
</ol>
<form name="Formular" action="">
<input type="button" name="abc" value="alphabetisch" onClick="ABC()">
<input type="button" name="geo" value="geographisch" onClick="GEO()">
</form>
</body></html>

```

Im Beispiel ist die nummerierte Liste zunächst mit den Städtenamen in alphabetischer Sortierfolge notiert. In dem Formular unterhalb der Liste sind zwei Klick-Buttons notiert. Der eine ruft die Funktion `ABC()` auf, der andere die Funktion `GEO()`. Ein Klick auf `ABC()` bringt natürlich nichts, da die Liste ohnehin alphabetisch sortiert ist. Also empfiehlt sich zunächst der Klick auf den Button mit der Aufschrift `geographisch`. Die Funktion `GEO()`, die er aufruft, definiert

zunächst mit `var Staedte = new Array();` einen neuen, leeren Array. Das Vorhaben ist, diesen Array mit den Städtenamen aus der Liste zu füllen. Dazu greift die Funktion der Reihe nach in einer `for`-Schleife alle Elemente vom Typ `li` ab, die in dem Dokument vorkommen.



21.1: Ein Klick auf die Buttons sortiert die Liste entsprechend um.

Der Zugriff erfolgt mit `document.getElementsByTagName("li")`. Über `document.getElementsByTagName("li").length` kann die Anzahl der `li`-Elemente im Dokument ermittelt werden. Diese Information benutzt die Funktion als Abbruchbedingung für die `for`-Schleife. Innerhalb der Schleife wird dem Array mit der `push`-Methode der Inhalt des jeweils aktuellen `li`-Elements hinzugefügt.

Mit `document.getElementsByTagName("li")[i].firstChild.nodeValue` wird dabei auf den Inhalt des jeweiligen `li`-Elements zugegriffen. Aus DOM-Sicht ist `document.getElementsByTagName("li")[i]` ein Knoten, nämlich der *i*-te `li`-Elementknoten im Dokument. `firstChild` ist eine Eigenschaft des `node`-Objekts. Über diese Eigenschaft können Sie auf den ersten Kindknoten eines Knotens zugreifen. Der erste Kindknoten aller `li`-Elemente im Dokument ist deren Textinhalt, also jeweils ein Städtenamen. Die Eigenschaft `firstChild` liefert aber noch nicht den Inhalt des Elements, sondern nur das Objekt des Inhalts. Um tatsächlich an den Inhalt heranzukommen, also an den konkreten Städtenamen, muss eine weitere Eigenschaft des `node`-Objekts bemüht werden, nämlich die Eigenschaft `nodeValue`. Die Kombination `firstChild.nodeValue` können Sie sich ruhig merken. Diese Kombination werden Sie häufig verwenden, um an den Inhalt eines Elements zu kommen.

Nachdem der Array mit den Städtenamen gefüllt ist, tut die Funktion `GE0()` einfach das Umgekehrte und schreibt den Array wieder zurück in die Liste – ebenfalls in einer `for`-Schleife. Der Ausdruck `document.getElementsByTagName("li")[i].firstChild.nodeValue` steht diesmal links von der Zuweisung. Dadurch wird dem Inhalt des Listenelements dynamisch ein neuer Wert zugewiesen. Im Beispiel ist das der etwas vertrackt aussehende Wert `Staedte[geoArray[i]]`. Die Städte sollen ja geographisch ausgegeben werden. Nun gibt es keinen Algorithmus,

der die Geographie kennt. Ganz oben im Script-Bereich ist daher ein Array namens `geoArray` notiert. Die Zahlen, mit denen er initialisiert wird, sind einfach die Indexnummern der alphabetisch sortierten Städte. So hat `geoArray[0]` beispielsweise den Wert 6. Dank dieser Information weiß die Funktion `GE0()`, dass die nördlichste der Städte diejenige ist, die in der alphabetischen Sortierung die Indexnummer 6 hat (Hamburg). Mit `Staedte[geoArray[i]]` benutzt die Funktion als aktuellen Index für die Stadt, die in die Liste geschrieben werden soll, also einfach den Zugriff auf `geoArray`.

Die Funktion `ABC()` geht ganz genauso vor wie die Funktion `GE0()`. Sie unterscheidet sich nur dadurch von Letzterer, dass sie nach dem Einlesen des `Staedte`-Arrays die Objekt-Methode `sort()` auf den Array anwendet, um die Einträge zu sortieren. Dann schreibt sie den sortierten Array einfach zurück in die nummerierte Liste.

Ein weiteres Element wird von beiden Funktionen ebenfalls noch geändert: nämlich das `span`-Element innerhalb der Überschrift. Da für dieses Element im HTML-Code mit `id="Art"` ein Id-Wert notiert ist, lässt sich mit `document.getElementById("Art")` bequem darauf zugreifen. Mit der üblichen Syntax `firstChild.nodeValue` wird der Text des Elements angesprochen und dynamisch geändert.

### 21.2.3 HTML-Attribute dynamisch ändern

Durch das dynamische Hinzufügen, Ändern oder Löschen von HTML-Attributen lassen sich interessante Effekte, aber auch nützliche Zwecke erreichen. Das folgende Beispiel zeigt, wie Sie Verweise dynamisch ändern können. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var TopLinks = new Array(
    "http://www.spiegel.de/",
    "http://www.heise.de/newsticker/",
    "http://www.encarta.msn.de/",
    "http://paperball.fireball.de/",
    "http://kochbuch.unix-ag.uni-kl.de/"
)
var aktuellerLink = 0;

function TopLink() {
    document.getElementsByName("Ziel")[0].href = TopLinks[aktuellerLink];
    document.getElementsByName("Ziel")[0].firstChild.firstChild.nodeValue =
TopLinks[aktuellerLink];
    aktuellerLink += 1;
    if(aktuellerLink >= TopLinks.length)
        aktuellerLink = 0;
}

function TopLinksFenster(Fenster) {
    document.getElementsByName("Ziel")[0].target = Fenster;
```

```

}
//-->
</script>
</head><body>
<p><a name="Ziel" href="#Ziel"><b>Top-Link?</b></a><br>
<a href="javascript:TopLink()">Nächster Top-Link!</a><br>
TopLinks: <a href="javascript:TopLinksFenster('_blank')">in neues Fenster
laden!</a> oder
<a href="javascript:TopLinksFenster('_self')">ins aktuelle Fenster laden!</a></p>
</body></html>

```

Das Beispiel enthält im sichtbaren Bereich einen »Top-Link«, der zunächst auf sich selbst verweist (name="Ziel" href="#Ziel"). Unterhalb davon sind drei weitere Verweise notiert. Der erste verspricht den jeweils nächsten Top-Link zu präsentieren, und die beiden darunter erlauben es dem Anwender festzulegen, wie er die Links öffnen will – ob im gleichen oder in einem neuen Fenster. Alle diese drei Verweise rufen JavaScript-Funktionen auf, die im Dateikopf notiert sind. Der Verweis, der den nächsten Top-Link verspricht, ruft die Funktion `TopLink()` auf.



21.2: Per Mausklick verändert sich der in der ersten Zeile angezeigte Link.

Diese Funktion greift mit `document.getElementsByName("Ziel")[0]` auf das erste Element im Dokument zu, bei dem als Attribut name="Ziel" notiert ist. Das ist im Beispiel bei dem Verweis der Fall, der die Top-Links anzeigen soll, aber bislang auf sich selbst verweist. Geändert werden sollen bei diesem Verweis sowohl sein href-Attribut als auch sein sichtbarer Verweistext. Der Zugriff auf das href-Attribut ist ganz einfach, weil nach der DOM-Syntax der HTML-Elementobjekte jedes denkbare Attribut eines HTML-Elements einfach eine Eigenschaft des entsprechenden HTML-Objekts darstellt. Um auf HTML-Attribute zuzugreifen, brauchen Sie also nur auf das gewünschte Element zuzugreifen und dahinter, durch Punkt getrennt, den Attributnamen als Eigenschaft notieren. Beachten Sie dabei aber die besondere Groß-/Kleinschreibung bei verschiedenen Attributen wie `bgColor`, `vSpace` oder `cellPadding`.

Mit `document.getElementsByName("Ziel")[0].href` wird im Beispiel also direkt auf den Wert des href-Attributs des gewünschten Verweises zugegriffen. Durch Zuweisung eines Wertes wird das Attribut dynamisch geändert. Im Beispiel wird jeweils ein neuer Wert aus dem weiter oben im Script notierten Array namens `TopLinks` zugewiesen. Als Indexzähler wird die ebenfalls zuvor schon definierte Variable `aktuellerLink` verwendet. Diese wird anschließend um 1 erhöht, damit beim nächsten Aufruf der Funktion der nächste Top-Link aus dem Array gesetzt wird. Wenn der Wert von `aktuellerLink` zu hoch wird, um noch einen Eintrag in dem Array



zu finden, wird die Variable wieder auf 0 gesetzt, und beim nächsten Klick wird wieder der erste Eintrag aus dem Array serviert.

Der Zugriff auf den Verweistext erfolgt wieder über `firstChild.value`. Allerdings ist im Beispiel `firstChild.firstChild.nodeValue` notiert. Der Grund ist, dass der Verweistext nicht direkt zwischen `<a>` und `</a>` steht, sondern nochmal in ein `b`-Element zur Fett-Auszeichnung eingeschlossen ist. Aus Sicht des `a`-Elements ist also das `b`-Element das `firstChild`, und erst dessen `firstChild` ist der sichtbare Verweistext.

Die Verweise, die regeln, in welchem Fenster die Top-Links geöffnet werden sollen, rufen jeweils die Funktion `TopLinksFenster()` auf und übergeben ihr als Parameter den gewünschten Zielfensternamen. Übergeben werden die reservierten Fensteramen `_self` (Öffnen im eigenen Fenster) bzw. `_blank` (Öffnen in neuem Fenster). Die Funktion `TopLinksFenster()` benutzt wieder den bequemen Weg über die HTML-Elementobjekte, um direkt auf die `target`-Eigenschaft des `a`-Elements zuzugreifen und das entsprechende Attribut zu setzen. Es ist also kein Problem, Attribute zu setzen, die im HTML-Quelltext eines Elements nicht notiert sind.

Falls Sie mal ein Attribut löschen wollen, haben Sie zwei Möglichkeiten: Entweder Sie setzen seinen Wert auf `""` (leere Zeichenkette), oder – und das ist sauberer – Sie arbeiten mit der Methode `removeAttribute()` des `node`-Objekts.

### 21.2.4 CSS-Eigenschaften dynamisch ändern

Seit der DOM-Version 2.0 wird auch geregelt, wie auf Stylesheet-Angaben zugegriffen wird. Dabei hat man sich weitgehend an das seinerzeit von Microsoft eingeführte `style`-Objekt angelehnt. Das folgende Beispiel zeigt, wie Sie mit Hilfe der DOM-Technik typisches, klassisches dynamisches HTML mit Bewegung positionierter Elemente schreiben. Dazu dient die Möglichkeit, Elemente mit Hilfe von CSS-Eigenschaften absolut zu positionieren. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var rp, bp, ich;
var rpGeschw = 10, bpGeschw = 20;
var rpGrad = 0, bpGrad = 0;
var rpX = 170, rpY = 170, bpX = 170, bpY = 170;
var rpRadius = 150, bpRadius = 150;

function Init() {
    rp = eval(document.getElementById("roterPunkt"));
    bp = eval(document.getElementById("blauerPunkt"));
    ich = eval(document.getElementById("ich"));
    rp.style.position = "absolute";
    rp.style.top = 20;
    rp.style.left = 320;
    bp.style.position = "absolute";
    bp.style.top = 320;
    bp.style.left = 320;
    ich.style.position = "absolute";
```

```

    ich.style.top = 110;
    ich.style.left = 90;
    ich.style.fontFamily = "Courier New,Courier";
    ich.style.fontSize = "72pt";
    ich.style.fontWeight = "bold";
    ich.style.color = "#009900";
    rpKreis();
    bpKreis();
}

function rpKreis() {
    rpGrad += rpGeschw/1000;
    if(rpGrad > 360) rpGrad = 0;
    rp.style.top = Math.round(rpY + (rpRadius * Math.cos(rpGrad)));
    rp.style.left = Math.round(rpX + (rpRadius * Math.sin(rpGrad)));
    window.setTimeout("rpKreis()",100/rpGeschw);
}

function bpKreis() {
    bpGrad += bpGeschw/1000;
    if(bpGrad > 360) bpGrad = 0;
    bp.style.top = Math.round(bpY + (bpRadius * Math.cos(bpGrad)));
    bp.style.left = Math.round(bpX + (bpRadius * Math.sin(bpGrad)));
    window.setTimeout("bpKreis()",100/bpGeschw);
}
//-->
</script>
</head><body onLoad="Init()">

<div id="roterPunkt"></div>
<div id="blauerPunkt"></div>
<div id="ich">ICH</div>

</body></html>

```

Innerhalb des `body`-Bereichs werden im Beispiel einfach nur drei `div`-Bereiche ohne weitere Formatierungen notiert. Die ersten beiden enthalten jeweils eine kleine Grafik – *ichkreis1.gif* ist ein roter Punkt, *ichkreis2.gif* ein blauer Punkt. Beide Grafiken haben einen transparenten Hintergrund, was wichtig ist, da sich die Punkte im späteren Verlauf des Geschehens öfter überlagern werden.

Im einleitenden `<body>`-Tag ist der Event-Handler `onLoad=` notiert. Dieser tritt in Aktion, sobald die Datei vollständig im Browser geladen ist. Dann wird die Funktion `Init()` aufgerufen, die im Dateikopf notiert ist. Diese Funktion setzt zunächst einmal die JavaScript-Funktion `eval()` ein, um den Zugriff auf die `div`-Bereiche zu verkürzen. Da alle drei `div`-Bereiche ein `id`-Attribut haben, ist der Zugriff auf den entsprechenden Elementknoten mit `getElement-`

ById("roterPunkt") usw. möglich. Durch Anwendung von `eval` auf einen solchen Ausdruck wird in einer Variablen wie `rp` quasi die »Bedeutung« von `getElementById("roterPunkt")` gespeichert, und anschließend lässt sich mit `rp` genauso arbeiten, als wenn man jedes Mal wieder `getElementById("roterPunkt")` notieren würde.



21.3: Das »Ich« umkreisen ein roter und ein blauer Punkt mit unterschiedlicher Geschwindigkeit.

Über die Variablen `rp`, `bp` und `ich` ist dann auch das `style`-Objekt ansprechbar. Die Funktion `Init()` stattet die drei `div`-Bereiche erst einmal mit anfänglichen CSS-Eigenschaften aus. Alle drei Bereiche werden mit `style.position = "absolute"` absolut positioniert. Die linke obere Ecke jedes Bereichs wird mit `style.left` und `style.top` bestimmt. Der Inhalt des `ich`-Bereichs, also das Wort `ICH`, wird groß und auffällig formatiert.

Am Ende ruft die Funktion `Init()` die beiden anderen Funktionen `rpKreis()` und `bpKreis()` auf. Jede dieser beiden Funktionen kontrolliert die Kreisbewegung jeweils eines der beiden Punkte, indem sie letztlich die linke obere Ecke des `div`-Bereichs, der die jeweilige Grafik enthält, neu berechnet. Dabei kommen die Kreisfunktionen für Sinus (`Math.sin()`) und Cosinus (`Math.cos()`) zum Einsatz. Am Ende ruft jede der beiden Funktionen sich selber rekursiv mit `window.setTimeout()` wieder auf, um die nächste Position des `div`-Bereichs mit der Grafik zu setzen. Die Variablen wie `bpGrad`, `bpGeschw` usw., mit denen diese Funktionen arbeiten, wurden zuvor im globalen Bereich des Scripts initialisiert.

### 21.2.5 Neue Knoten erzeugen und in Baumstruktur einhängen

Dynamisches HTML nach DOM bedeutet nicht nur, auf bereits in HTML vorhandene Elemente zuzugreifen, sondern auch neue zu erzeugen und in den Strukturbaum einzuhängen. Wie das funktioniert, zeigt das folgende Beispiel. Es stellt einen primitiven HTML-Editor dar. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
```

```

h1 { font-family:Arial,sans-serif; font-size:24pt; font-weight:normal; border-
bottom:2px solid red; }
h2 { font-family:Arial,sans-serif; font-size:20pt; font-weight:normal; border-
bottom:1px solid red; }
h3 { font-family:Arial,sans-serif; font-size:12pt; font-weight:bold; }
p { font-family:Arial,sans-serif; font-size:11pt; }
-->
</style>
<script type="text/javascript">
<!--
function Hinzufuegen() {
    var Typ =
document.Formular.Elementtyp.options[document.Formular.Elementtyp.selectedIndex].va
lue;
    var Elementknoten = document.createElement(Typ);
    if(Typ == "hr")
        document.getElementById("User").appendChild(Elementknoten);
    else {
        var Textknoten =
document.createTextNode(document.Formular.Elementinhalt.value);
        Elementknoten.appendChild(Textknoten);
        document.getElementById("User").appendChild(Elementknoten);
    }
}
//-->
</script>
</head><body>

<form name="Formular" style="background-color:#EEEEEE; padding:10px" action="">
<select name="Elementtyp" size="1">
<option value="h1">Überschrift 1</option>
<option value="h2">Überschrift 2</option>
<option value="h3">Überschrift 3</option>
<option value="p">Textabsatz</option>
<option value="hr">Trennlinie</option>
</select>
<input type="text" name="Elementinhalt" size="50">
<input type="button" value="Hinzufügen" onClick="Hinzufuegen()">
</form>

<div id="User" style="background-color:#FFFFC0; padding:10px">
</div>

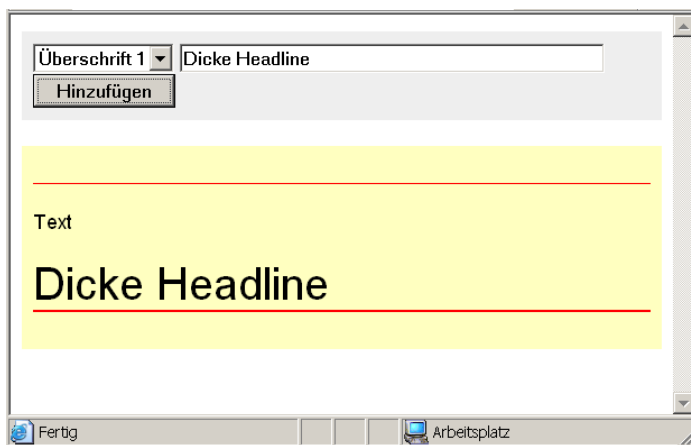
</body></html>

```

Das Beispiel enthält im sichtbaren HTML-Bereich ein Formular mit einer Auswahlliste, einem Eingabefeld und einem Klick-Button. Unterhalb ist noch ein `div`-Bereich notiert, der jedoch noch keinen konkreten Inhalt hat. In der Auswahlliste des Formulars kann der Anwender einen

HTML-Elementtyp auswählen – in den `value`-Attributen der `option`-Elemente sind die tatsächlichen HTML-Elementnamen der jeweiligen Auswahl gespeichert. Im Eingabefeld des Formulars kann der Anwender den Textinhalt für das ausgewählte Element eingeben. Wenn er dann auf den Button klickt, wird die Funktion `Hinzufuegen()` aufgerufen, die im Dateikopf in einem JavaScript-Bereich notiert ist.

Dort wird erst einmal der lange, umständliche Ausdruck `document.Formular.Elementtyp.options[document.Formular.Elementtyp.selectedIndex].value` in der Variablen `Typ` gespeichert. Der lange Ausdruck bzw. die Variable speichert den vom Anwender im Formular ausgewählten HTML-Elementtyp, also z.B. `h1` oder `p`.



21.4: Auf Mausclick ergänzt das Script die Web-Seite um zusätzliche Elemente.

Mit `document.createElement(Typ)` wird dann ein leeres, neues Element vom Typ `Typ` erzeugt, also je nach Wert der Variablen `Typ` z.B. ein `h1`-Element oder ein `p`-Element. Damit wird das Element aber noch nicht angezeigt. `document.createElement()` erzeugt lediglich den Elementknoten, hängt ihn aber noch nicht in den Strukturbaum des Dokuments ein. Ebenso ist es mit `document.createTextNode()`. Diese Methode erzeugt einen Textknoten, aber legt in keiner Weise fest, wohin dieser Textknoten gehört. Im Beispiel wird aus dem, was der Anwender im Eingabefeld des Formulars eingegeben hat, mit `document.createTextNode(document.Formular.Elementinhalt.value)` ein solcher Textknoten erzeugt. Das »Einhängen« in den Strukturbaum ist die nächste Aufgabe des Scripts.

Dazu muss das Script jedoch zwischen Elementen unterscheiden, die einen Textinhalt haben, und solchen, die keinen haben. Das `hr`-Element, das der Anwender ebenfalls auswählen kann, also das HTML-Element für Trennlinien, kann keinen Textinhalt haben. Alle anderen auswählbaren Elemente dagegen können Textinhalt haben. Deshalb unterscheidet das Script mit `if` und `else` zwischen `Typ == "hr"` und allen übrigen möglichen Werten von `Typ`. Wenn der Anwender das `hr`-Element ausgewählt hat, wird einfach nur das neu erzeugte und in der Variablen `Elementknoten` gespeicherte `hr`-Element in den Dokumentbaum eingehängt. Der Textknoten wird dagegen verworfen. In allen anderen Fällen (`else`-Zweig) muss zunächst der Textknoten in den Elementknoten eingehängt werden, damit das Element den Text als Inhalt erhält. Anschließend muss das Element in den Dokumentbaum eingehängt werden.

Für alle »Einhäng«-Vorgänge wird die Methode `appendChild()` des `node`-Objekts verwendet. Anwendbar ist die Methode auf ein Knotenobjekt, das Kindknoten haben darf, also beispielsweise Elementknoten. Als Parameter erwartet die Methode einen Knoten, der als Kindknoten eingehängt werden soll. Wenn im Beispiel also steht:

```
Elementknoten.appendChild(Textknoten);
```

dann ist `Elementknoten` eine Variable, in der zuvor durch `createElement()` ein Elementobjekt erzeugt wurde, und `Textknoten` ist eine Variable, die zuvor durch Aufruf von `createTextNode()` einen Textknoten gespeichert hat. Mit der Anweisung `document.getElementById("User").appendChild(Elementknoten)` wird auf den zunächst leeren `div`-Bereich im Dokument zugegriffen. Diesem Element wird der neu erzeugte Elementknoten hinzugefügt.

## 21.2.6 Event-Handling beim DOM

Die Normierung des Event-Handling im DOM war zum Redaktionszeitpunkt dieses Dokuments noch nicht abgeschlossen – so fehlte beispielsweise noch die Implementierung von Tastaturereignissen. Der Internet Explorer 5.x interpretiert auch noch keine Events nach DOM-Syntax, Netscape 6.x dagegen schon. Das folgende Beispiel zeigt das Prinzip, nach dem Event-Handling nach DOM-Syntax funktioniert. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Ausgeben(Text) {
    LogEintrag = document.createElement("li");
    neuerText = document.createTextNode(Text);
    LogEintrag.appendChild(neuerText);
    document.getElementById("Log").appendChild(LogEintrag);
}

function handleEingabefeldClick(ev) {
    Event = ev;
    Text = "Sie haben in das Eingabefeld geklickt. ";
    Text = Text + "Event-Typ = " + Event.type + ", ";
    Text = Text + "Event-Target-Elementname = " + Event.target.nodeName + ".";
    Ausgeben(Text);
}

function handleBereichMove(ev) {
    Event = ev;
    Text = "Sie haben die Maus im Bereich bewegt. ";
    Text = Text + "X-Position = " + Event.clientX + ", ";
    Text = Text + "Y-Position = " + Event.clientY + ".";
    Ausgeben(Text);
}

function handleBereichClick(ev) {
```

```

    Event = ev;
    Text = "Sie haben in den Bereich geklickt. ";
    Text = Text + "X-Position = " + Event.clientX + ", ";
    Text = Text + "Y-Position = " + Event.clientY + ".";
    Ausgeben(Text);
}

function Init() {
    document.getElementsByName("Eingabe")[0].addEventListener("click",
handleEingabefeldClick, true);
    document.getElementById("Bereich").addEventListener("mousemove",
handleBereichMove, true);
    document.getElementById("Bereich").addEventListener("click", handleBereichClick,
true);
}
//-->
</script>
</head><body onLoad="Init()">

<form name="Formular" style="background-color:#EEEEEE; padding:10px" action="">
<input type="text" name="Eingabe" size="50">
</form>

<div id="Bereich" style="background-color:#FF0000; color:#FFFFFF; font-weight:bold;
width:100px">Ein Bereich</div>

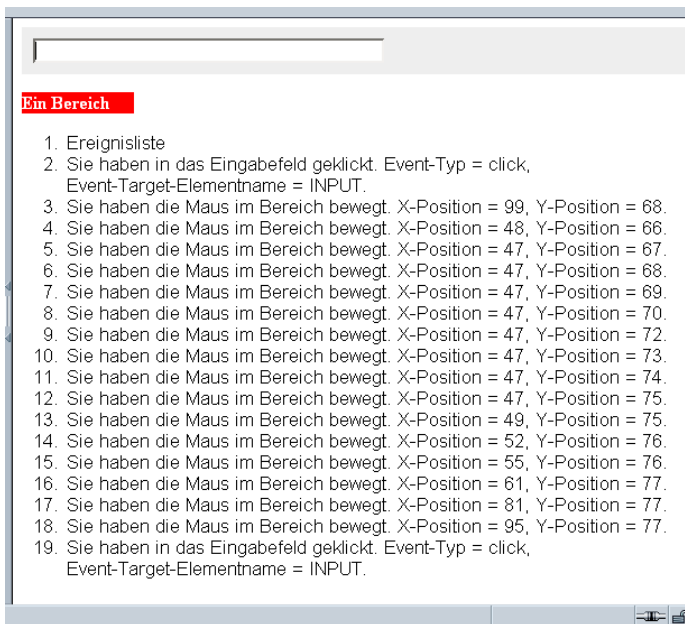
<ol id="Log" style="font-family:Arial,sans-serif; font-
size:10pt"><li>Ereignisliste</li></ol>

</body></html>

```

Im sichtbaren Bereich der HTML-Datei ist ein Formular mit einem Eingabefeld notiert, ferner ein `div`-Bereich, und schließlich eine leere, nummerierte Liste. Im einleitenden `<body>`-Tag ist der Event-Handler `onLoad` notiert, der jedoch noch nichts mit Event-Handling nach DOM-Syntax zu tun hat. Dort wird einfach erst mal die Funktion `Init()` aufgerufen, die im Dateikopf notiert ist.

Innerhalb der Funktion `Init()` geht es jedoch in Sachen DOM-Event-Handling zur Sache. Event-Handling besteht beim DOM zunächst einmal darin, für einen beliebigen Knoten im Dokument eine Ereignisüberwachung zu registrieren. Dazu gibt es die Methode `addEventListener()`. Mit dem Teil davor, im Beispiel etwa `document.getElementsByName("Eingabe")[0]`, wird derjenige Knoten im Dokument angesprochen, für den eine Ereignisüberwachung registriert werden soll. Im Beispiel der Funktion `Init()` werden insgesamt drei Ereignisüberwachungen angestoßen: eine für das Eingabefeld des Formulars, und zwei für den `div`-Bereich.



21.5: Die Liste im unteren Bereich zeigt in Netscape 6 die ausgelösten Ereignisse.

Die Methode `addEventListener()` erwartet drei Parameter. Der erste Parameter gibt an, welcher Ereignistyp überwacht werden soll. Bei Mausereignissen sind das weitgehend die aus dem JavaScript-Objekt `event` bekannten Ereignistypen wie `click`, `mouseover`, `mousedown`, `mouseup`, `mousemove`. Dazu kommen im DOM Event-Typen wie `DOMFocusIn` (Knoten erhält den Fokus), `DOMFocusOut` (Knoten verliert den Fokus), `DOMActivate` (Knoten wird durch Mausklick oder Tastendruck aktiviert), sowie anwenderunabhängige Ereignisse wie `DOMSubtreeModified` (Strukturbaum geändert), `DOMNodeInserted` (Knoten in Strukturbaum eingefügt) oder `DOMNodeRemoved` (Knoten aus Strukturbaum entfernt). Der Name des gewünschten Ereignistyps muss bei der Parameterangabe in Anführungszeichen stehen.

Der zweite Parameter, den `addEventListener()` erwartet, ist der Name einer Funktion, die beim Eintreten des Events aufgerufen werden soll. Der Name der Funktion wird ohne Anführungszeichen angegeben. Die angegebene Funktion bekommt automatisch ein Ereignisobjekt übergeben und kann dann damit anfangen, was sie will. Dazu weiter unten.

Als dritter Parameter muss `addEventListener()` entweder `true` oder `false` übergeben werden. Mit `true` schalten Sie die Event-Überwachung ein, mit `false` noch nicht.

Für jedes Ereignis, das Sie überwachen wollen, benötigen Sie eine Funktion, die auf das Ereignis reagiert. Es ist die Funktion mit dem Namen, der bei `addEventListener()` als zweiter Parameter angegeben wird. Im obigen Beispiel sollen drei Ereignisse überwacht werden. Deshalb gibt es drei ähnliche Funktionen – mit den Namen `handleEingabefeldClick()`, `handleBereichMove()` und `handleBereichClick()`. Alle drei Funktionen bekommen beim automatischen Aufruf, der beim Eintritt des Ereignisses passiert, das Ereignis als Objekt übergeben. Die Funktionen im Beispiel erwarten deshalb einen Parameter `ev`, den sie in ihrer jeweils ersten Anweisung der Variablen `Event` zuweisen. Danach können die Funktionen über die `Event`-Variable auf



Eigenschaften und Methoden des Ereignisobjekts zugreifen. Bei Mausereignissen gibt es beispielsweise Eigenschaften wie `clientX` (Anzahl Pixel vom linken Rand des sichtbaren Knotenbereichs), `clientY` (Anzahl Pixel vom oberen Rand des sichtbaren Knotenbereichs), `screenX` (Anzahl Pixel vom linken Bildschirmrand) oder `screenY` (Anzahl Pixel vom oberen Bildschirmrand).

Im Beispiel stellen die Funktionen, die ein Ereignis überwachen, einfach nur einen Text zusammen und rufen dann die Funktion `Ausgeben()` mit dem zusammengestellten Text auf. Die Funktion `Ausgeben()` erzeugt zuerst mit `document.createElement("li")` ein neues HTML-Element vom Typ `li`. Dann erzeugt sie mit `document.createTextNode(Text)` einen neuen Textknoten mit dem übergebenen Text. Schließlich hängt sie den Textknoten mit `appendChild()` als Kindknoten in den erzeugten `li`-Elementknoten ein, und diesen in die im Dokument notierte nummerierte Liste. Auf diese Weise füllt sich die nummerierte Liste dynamisch mit jedem Ereignis, das erkannt wurde.

## 21.3 Das ältere DHTML-Modell von Microsoft

### 21.3.1 Allgemeines zum Microsoft-Modell

Als der MS Internet Explorer 4.0 im Jahr 1997 ein völlig neues Konzept mit ganz neuen Möglichkeiten des dynamischen Zugriffs auf eine Web-Seite einführte, waren die meisten Web-Entwickler noch gänzlich in dem klassischen JavaScript-Ansatz von Netscape befangen und begriffen gar nicht so recht die Tragweite des neuen Modells. Das Modell erlaubte nichts Geringeres, als beliebige HTML-Elemente inklusive deren CSS-Eigenschaften zu ändern, auszutauschen, neu zu erzeugen, zu löschen usw. Der Browser reagierte dabei wie eine Textverarbeitungssoftware während der Bearbeitung. Wenn Inhalte dynamisch eingefügt wurden, wurde alles, was dahinter stand, automatisch neu positioniert. Obwohl diese Fähigkeit – eben im Hinblick auf moderne Textverarbeitungssoftware – im Grunde genommen von einer professionellen Software zu erwarten ist, war dem zu jener Zeit nicht so. Der 1997 noch dominante Netscape-Browser war genau dazu nicht in der Lage: eine Seite aufgrund dynamischer Änderungen wirklich neu aufzubauen. Auch die Web-Entwickler begriffen lange Zeit nicht, dass es doch viel einfacher und konsequenter ist, beliebige HTML-Elemente, deren Eigenschaften und Inhalte dynamisch zu ändern, anstatt mit der umständlichen Layer-Technik von Netscape zu hantieren.

Das Microsoft-Modell war also richtungsweisend. Aber es war proprietär, ein firmeneigener Vorstoß in neue Welten ohne Absprache mit einem unabhängigen Normierungs-Gremium. Microsoft benahm sich in jener Zeit in der Tat unglaublich aggressiv und versuchte, die verpennte Zeit der Jahre davor aufzuholen. Immerhin hatte Bill Gates noch ganze zwei Jahre zuvor das Internet als unwichtig und für Microsoft bedeutungslos erklärt. Der Versuch, die »Sahnetorte Internet« mit aggressiver Marketing-Politik alleine aufzufressen, gelang jedoch glücklicherweise nur zum Teil. Zwar eroberte der Internet Explorer Monat um Monat mehr Anteile am Browser-Markt und ist mittlerweile längst der am weitesten verbreitete Browser. Doch Microsofts Versuch, eigene Sprachen, Formate und Programmiertechniken zu etablieren, ist fehlgeschlagen. In diesem Punkt reagierte die Web-Gemeinde empfindlich. Man hatte erkannt, dass Software vor allem dann von sich abhängig macht, wenn sie Dateiformate diktiert.

Das DHTML-Modell von Microsoft war also ein wichtiger Meilenstein auf dem Weg zum heute zu empfehlenden DOM. Das DOM weicht aber insofern von dem Microsoft-Ansatz ab, als es nicht auf HTML fixiert ist, sondern gleich alle XML-kompatiblen Tag-Sprachen abdeckt. Der Microsoft-Ansatz ist aus heutiger Sicht nur noch von Bedeutung, wenn Sie dynamische Webseiten erstellen wollen, die auch mit dem Internet Explorer 4.x darstellbar sein sollen. Zwar verschwindet dessen Anteil am Browser-Markt immer weiter gegenüber den neueren Versionen, die seit der 5er-Version DOM-fähig sind, doch Web-Design sollte wann immer es möglich ist auch auf ältere Browser-Versionen Rücksicht nehmen.

### 21.3.2 Script-Sprachen: JavaScript, JScript und VBScript

Es gibt letztlich keine Vorschriften dafür, welche Script-Sprache für Dynamisches HTML verwendet werden soll. Deshalb herrscht hierbei noch etwas Verwirrung.

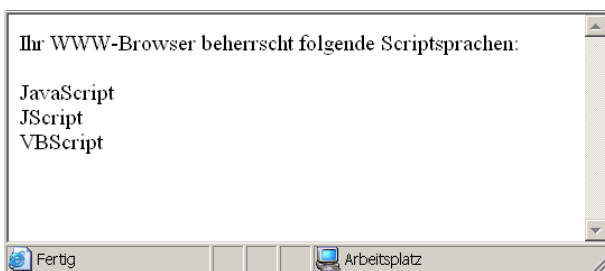
JavaScript ist jedoch eindeutig die meist verwendete und bekannteste Script-Sprache. Sie wurde durch Netscape eingeführt. Der MS Internet Explorer interpretiert JavaScript ebenfalls. Dabei hinkt er Netscape jedoch immer etwas hinterher, da Netscape die Lizenzen für JavaScript besitzt und es Microsoft nicht erlaubt ist, den Sprachumfang von JavaScript selbstständig zu erweitern. Mit einem kleinen Kniff hat sich Microsoft aus dieser misslichen Lage befreit: die Sprache wurde einfach in JScript umbenannt. JScript ist JavaScript plus MS-eigene Spracherweiterungen. JScript enthält etliche Objekte, Methoden und Eigenschaften, die in JavaScript nicht vorkommen. Die meisten dieser Befehle sind für die lokalen Betriebssystemerweiterungen gedacht, die der MS Internet Explorer 4.x unter MS Windows benötigt. Aber auch das `all`-Objekt, das in Netscapes JavaScript nicht vorkommt, gehört dazu. Und dieses Objekt ist ganz entscheidend für Dynamisches HTML im MS Internet Explorer.

Neben JavaScript und JScript beherrscht der MS Internet Explorer auch noch VBScript. Diese Sprache ist von Visual Basic (VB) abgeleitet. Sie konnte sich bislang jedoch nicht durchsetzen im WWW. VBScript hat eine andere Syntax als JavaScript und JScript. Die Objekte, Eigenschaften und Methoden sind jedoch im Wesentlichen die gleichen. Mit VBScript können Sie also ebenfalls Dynamisches HTML schreiben. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<p>Ihr WWW-Browser beherrscht folgende Script-Sprachen:<br>
<script language="JavaScript" type="text/javascript">
<!--
document.write(<br>JavaScript");
//-->
</script>
<script language="JScript" type="text/jscript">
<!--
document.write(<br>JScript");
//-->
</script>
<script language="VBScript" type="text/vbscript">
<!--
document.write "<br>VBScript"
```

```
//-->
</script>
</p>
</body></html>
```

Das obige Beispiel ist kein Dynamisches HTML im engeren Sinn. Es soll lediglich demonstrieren, wie Sie unterschiedliche Script-Sprachen einsetzen können, und welche Wirkung dies auf WWW-Browser hat. In dem Beispiel werden drei Script-Bereiche definiert: einer mit JavaScript (`<script language="JavaScript" type="text/javascript">`), einer mit JScript (`<script language="JScript" type="text/jscript">`) und einer mit VBScript (`<script language="VBScript" type="text/vbscript">`). Jeder der drei Bereiche schreibt mit Hilfe eines `document.write`-Befehls (dieser Befehl ist in allen drei Sprachen gleich) eine Zeile in das Dokument. Da ein WWW-Browser nur diejenigen Script-Bereiche ausführt, deren Sprachen er kennt, stehen am Ende genau diejenigen Script-Sprachen auf der Seite, die der benutzte WWW-Browser kennt.



21.6: Wenig überraschend: IE6 unterstützt die drei genannten Script-Sprachen.

Sie sollten einen Script-Bereich einfach dann mit »JScript« auszeichnen, wenn Sie verhindern wollen, dass Netscape den Bereich interpretiert. Der MS Internet Explorer 4.x selbst nimmt es andererseits nicht übel, wenn Sie einen Script-Bereich mit »JavaScript« auszeichnen und darin Objekte, Methoden und Eigenschaften verwenden, die eigentlich nur in JScript vorkommen. Wenn Sie einen Script-Bereich mit »JavaScript« auszeichnen und darin Microsoft-spezifische JScript-Erweiterungen wie das `all`-Objekt verwenden, sollten Sie die Ausführung entsprechender Anweisungen von einer Abfrage abhängig machen. Lesen Sie dazu das Thema *Abfragen ob ein Objekt existiert* in JavaScript (Kapitel 18.3.6).

**Beachten Sie:** In modernen Browsern ist es normalerweise ausreichend, die Eigenschaft `type=` anzugeben. Ältere Browser und Opera 5.12 benötigen jedoch zwingend das Attribut `language=`, um den Script-Bereich von der Bearbeitung auszuschließen.

### 21.3.3 Das all-Objekt

Das `all`-Objekt ist beim DHTML-Ansatz von Microsoft der entscheidende Schlüssel. Wie der Name schon verspricht, ist über dieses Objekt jedes einzelne HTML-Element einer HTML-Datei ansprechbar. Das `all`-Objekt hat etliche Eigenschaften und Methoden, die das dynamische Verändern eines HTML-Elements erlauben. Alle Methoden und Eigenschaften können an dieser Stelle nicht behandelt werden. Blättern Sie dazu die JavaScript-Objektreferenz in Kapitel 20.7 auf und verschaffen Sie sich einen Überblick über die Eigenschaften und Methoden des Objekts `all`.

Hier soll lediglich ein weiteres Beispiel die Wirkungsweise dieses Objekts verdeutlichen. Ein Beispiel:

```
<html>
<head><title>Test</title>
<script language="JScript" type="text/javascript">
<!--
  var Wochentagname =
  new
Array("Sonntag","Montag","Dienstag","Mittwoch","Donnerstag","Freitag","Samstag");
function ZeitAnzeigen()
{
  var Jetzt = new Date();
  var Tag = Jetzt.getDate();
  var Monat = Jetzt.getMonth() + 1;
  var Jahr = Jetzt.getYear();
  var Stunden = Jetzt.getHours();
  var Minuten = Jetzt.getMinutes();
  var Sekunden = Jetzt.getSeconds();
  var WoTag = Jetzt.getDay();
  var Vortag = ((Tag < 10) ? "0" : "");
  var Vormon = ((Monat < 10) ? ".0" : ".");
  var Vorstd = ((Stunden < 10) ? "0" : "");
  var Vormin = ((Minuten < 10) ? ":0" : ":");
  var Vorsek = ((Sekunden < 10) ? ":0" : ":");
  var Datum = Vortag + Tag + Vormon + Monat + "." + Jahr;
  var Uhrzeit = Vorstd + Stunden + Vormin + Minuten + Vorsek + Sekunden;
  document.all.Uhr.innerHTML = Datum + " (" + Wochentagname[WoTag] + ")<br>" +
Uhrzeit;
  window.setTimeout("ZeitAnzeigen()",1000);
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF">
<p id="Uhr" style="font-family:Verdana; font-size:24pt;
font-weight:bold; color:#CC00CC"></p>
<script language="JScript" type="text/javascript">
<!--
window.setTimeout("ZeitAnzeigen()",1000);
//-->
</script>
</body>
</html>
```



21.7: Die Uhr tickt – mit Hilfe des `all`-Objekts.

Das Beispiel stellt eine vollständige, laufende Uhr mit Datum dar. Dazu wird ein leerer Textabsatz mit `<p>...</p>` definiert. Stylesheet-Angaben dienen dazu, den Textabsatz nach Wunsch zu formatieren. Obwohl der Absatz zunächst keinen Text enthält, dient er im Verlauf des Scripts als Container (Behälter) für den angezeigten Text mit Datum und Uhrzeit.

Das obere Script ermittelt Datum und Uhrzeit und formatiert es für eine ordentliche, hierzulande übliche Schreibweise. Die Zeichenkette, die ausgegeben werden soll, wird schließlich der Objekteigenschaft `document.all.Uhr.innerHTML` zugewiesen. `Uhr` ist dabei der Bezug auf den Namen, der bei der Definition des Textabsatzes mit `id=` definiert wird. Mit der Objekteigenschaft `innerHTML` können Sie den gesamten Inhalt eines Elements, hier den des benannten `<p>...</p>`-Elements, dynamisch ersetzen. Da sich die Funktion zur Ausgabe der Zeit jede Sekunde selbst von neuem aufruft, wird jede Sekunde ein neuer Inhalt in das `<p>`-Tag mit der `id="Uhr"` geschrieben. Auf diese Weise entsteht der Effekt einer laufenden Uhr.

Es gibt weitere, mächtige Möglichkeiten des `all`-Objekts. Diese werden in der JavaScript-Objektreferenz beim Objekt `all` beschrieben. Hier nur noch einige typische Befehlsfolgen:

Angenommen, Sie haben folgendes HTML-Element, ein gewöhnlicher Textabsatz:

```
<p id="derText">Das ist Text in einem Textabsatz</p>
```

Dann können Sie beispielsweise folgende dynamische Veränderungen bewirken:

- Mit `document.all.derText.innerHTML = "<b>neuer Text</b>"` im gleichen Textabsatz; ändern Sie den Inhalt des Textabsatzes und können dabei auch HTML-Tags zur Formatierung benutzen.
- Mit `document.all.derText.outerHTML = "<h1>" + document.all.derText.innerHTML + "</h1>"`; verwandeln Sie den Textabsatz in eine Überschrift 1. Ordnung.
- Mit `document.all.derText.className = "Wichtig"`; weisen Sie dem Absatz eine Stylesheet-Klasse namens `Wichtig` zu. Wenn Sie eine solche Klasse definiert haben, übernimmt der Absatz alle in der Klasse definierten Stylesheet-Angaben (lesen Sie dazu auch den Abschnitt 15.3.5 *Formate für Klassen definieren*).

### 21.3.4 CSS Stylesheets und Scripts

Sie können mit Dynamischem HTML nach Microsoft-Syntax dynamisch auf alle CSS-Eigenschaften einzelner HTML-Elemente zugreifen. Dadurch können Sie angezeigte Elemente optisch stark verändern. Wenn etwa eine Überschrift 2. Ordnung anfangs eine Schriftgröße von 14

Punkt hat, können Sie diese dynamisch auf 42 Punkt ändern. »Dynamisch« bedeutet dabei: bei einem Anwenderereignis wie einem Mausklick auf die Überschrift oder automatisch zeitgesteuert, etwa nach drei Sekunden.

Neben der Schriftgröße können Sie auch dynamisch eine Hintergrundfarbe unterlegen, Rahmen um ein Element ziehen, das Element über den Bildschirm wandern lassen und vieles mehr. Die Voraussetzung für diese Effekte sind CSS Stylesheets. Die Syntax der CSS-Stylesheets und die Möglichkeiten der Stylesheet-Angaben sollten Sie kennen. Die Style-Definitionen eines HTML-Elements ändern Sie dynamisch, indem Sie mit Hilfe einer Script-Sprache auf das Element zugreifen und mit neuen Stylesheet-Angaben das Aussehen des Elements ändern. Ein Beispiel:

```
<html><head><title>Test</title></head><body>
<h1 id="Wechseltext" style="font-size:14pt">Wichtigstuer</h1>
<script language="JScript" type="text/jscript">
<!--
function MachKlein()
{
    document.all.Wechseltext.style.fontSize = "14pt";
    window.setTimeout("MachGross()",1000);
}
function MachGross()
{
    document.all.Wechseltext.style.fontSize = "42pt";
    window.setTimeout("MachKlein()",1000);
}
MachKlein();
//-->
</script>
</body></html>
```

Das Beispiel zeigt zunächst eine Überschrift 1. Ordnung (<h1>...</h1>). Mit Hilfe des Attributs style= bekommt die Überschrift CSS-Eigenschaften zugewiesen. Im Beispiel wird die Schriftgröße auf 14 Punkt gesetzt (font-size:14pt).

Das gesamte Beispiel bewirkt, dass die Größe der Überschrift einmal pro Sekunde von 14 auf 42 Punkt wechselt und wieder umgekehrt. Für dieses dynamische Verändern ist ein Script erforderlich. Im obigen Beispiel wird deshalb ein Script-Bereich mit <script language="JScript" type="text/jscript">...</script> definiert. Innerhalb des Script-Bereichs werden zwei Funktionen definiert. Die eine Funktion, MachKlein(), ändert die Schriftgröße der Überschrift auf 14 Punkt, die andere Funktion, MachGross(), auf 42 Punkt. Nach jeweils einer Sekunde (1000 Millisekunden) ruft jede der beiden Funktionen die jeweils andere wieder auf. Dadurch entsteht der Wechsel-Effekt.

Um das dynamische Verändern der Schriftgröße zu verstehen, müssen Sie folgende Zusammenhänge beachten: Im Überschriften-Tag <h1> wird mit dem Attribut id= ein Name, nämlich Wechseltext, zugewiesen. Diesen Namen können Sie frei vergeben. Er darf nur aus Buchstaben, Ziffern und Unterstrichen »\_« bestehen und darf keine Leerzeichen oder deutschen Umlaute enthalten. Das Script im Beispiel greift mit den Anweisungen, in denen document.all.Wechseltext vorkommt, auf genau dieses Element zu. Mit dem all-Objekt (document.all) werden alle Elemente einer HTML-Datei angesprochen. Durch den Namen dahinter wird genau

das Element angesprochen, das diesen Namen hat – im Beispiel `Wechseltext`. Dahinter folgt das Style-Objekt (`style`), das dem `all`-Objekt untergeordnet ist (Einzelheiten zu diesem Objekt erfahren Sie in der JavaScript-Objektreferenz bei `style`). Über das Style-Objekt können Sie als Eigenschaft eine erlaubte CSS-Stylesheet-Angabe notieren. Dabei gelten besondere Schreibweisen in JavaScript/JScript, die im Folgenden behandelt werden.

### 21.3.5 Schreibweise von CSS-Eigenschaften in JavaScript/JScript

Wenn Sie Stylesheet-Angaben innerhalb von Script-Sprachen verwenden, müssen Sie die Syntax der Script-Sprache berücksichtigen. Ein Problem beim Mischen von CSS Stylesheets und JavaScript oder JScript ist es, dass viele Stylesheet-Angaben Bindestriche im Namen haben, wie etwa `font-size`, `margin-top` oder `background-color`. Wenn Sie aus solchen Namen in JavaScript oder JScript einen Befehl wie:

```
document.all.MeinText.style.font-size = "24pt";
```

erzeugen, erhalten Sie eine Fehlermeldung. Der Grund ist, dass Bindestriche keine zulässigen Zeichen in Namen sind (der Bindestrich ist in diesen Sprachen ja der Operator für Subtraktionen).

Um dieses Problem zu umgehen, hat Microsoft eine einfache Regel aufgestellt. Wenn in einem Script eine Stylesheet-Angabe ausgelesen oder verändert werden soll, entfällt der Bindestrich, und der erste Buchstabe des Wortes hinter dem Bindestrich wird großgeschrieben. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
var modus = 1;
function Textspiel() {
  if(modus == 1) {
    document.all.MeinText.style.fontSize = "24pt";
    modus = 2;
    return;
  }
  if(modus == 2) {
    document.all.MeinText.style.padding = "8pt";
    document.all.MeinText.style.backgroundColor = "#FFFF00";
    modus = 3;
    return;
  }
  if(modus == 3) {
    document.all.MeinText.style.border = "8pt groove #FFCC33";
    document.all.MeinText.style.backgroundImage = "url(back.gif)";
    modus = 0;
    return;
  }
  if(modus == 0) {
```

```

document.all.MeinText.style.fontSize = "12pt";
document.all.MeinText.style.padding = "0pt";
document.all.MeinText.style.backgroundColor = "#FFFFFF";
document.all.MeinText.style.border = "none";
document.all.MeinText.style.backgroundImage = "none";
modus = 1;
return;
}
}
//-->
</script>
</head><body>
<p id="MeinText" style="font-size:12pt; background-color:#FFFFFF; padding:0pt;
border:none; background-image:none; cursor:hand" onClick="Textspiel()">Klicken Sie
auf diesen Text</p>
</body></html>

```

In dem Beispiel wird am Ende ein Textabsatz mit diversen Stylesheet-Angaben definiert. Einige dieser Stylesheet-Angaben enthalten Bindestriche, nämlich `font-size`, `background-color` und `background-image`. Innerhalb des Scripts, das über dem Textabsatz notiert ist, werden diese und andere Stylesheet-Angaben dynamisch verändert. Sie können erkennen, dass `font-size` zu `fontSize` wird, `background-color` zu `backgroundColor` und `background-image` zu `backgroundImage` – getreu der Regel.



21.8: Der Absatz ändert mit jedem Mausklick seine Gestalt.

Wenn Sie innerhalb eines Scripts einer Stylesheet-Eigenschaft einen neuen Wert zuweisen wollen, müssen Sie zuerst das gewünschte HTML-Objekt mit der `style`-Eigenschaft notieren, im obigen Beispiel etwa `document.all.MeinText.style.fontSize`. Dahinter folgt ein Gleichheitszeichen für die Zuweisung, und dahinter der Wert, den Sie zuweisen wollen, also etwa `"24pt"`. Dieser Wert muss bei JavaScript und JScript immer in Anführungszeichen stehen.

Diese Art der Wertzuweisung an eine Stylesheet-Angabe unterscheidet sich von der Art, wie Werte zu Stylesheet-Angaben bei normalen Stylesheet-Definitionen zugewiesen werden. Dort notieren Sie anstelle eines Gleichheitszeichens einen Doppelpunkt, so wie in den Stylesheet-Definitionen im Textabsatz des obigen Beispiels. Anführungszeichen sind dabei nicht nötig.

Das Script im Beispiel verändert die Formatierung des Textes innerhalb des Textabsatzes, wenn der Anwender mit der Maus auf den Text klickt. Immer, wenn dieses Ereignis eintritt, wird mit Hilfe des Event-Handlers `onClick=` die Funktion `Textspiel()` innerhalb des Scripts aufgeru-



fen. Das Script verwaltet eine global definierte Variable namens `modus`. Abhängig davon, welchen Wert diese Variable hat, werden dem Text diverse neue Stylesheet-Angaben zugewiesen. Das dynamische Verändern des Textabsatzes im Script wird durch die Angabe `id="MeinText"` im Textabsatz und Befehlen des Typs `document.all.MeinText.style` bewerkstelligt.

### 21.3.6 Event-Handler und das Event-Objekt

Eine wichtige Voraussetzung für Dynamisches HTML ist in vielen Fällen die Interaktion zwischen Anwender und angezeigter HTML-Datei. Die angezeigte Datei kann auf Anwenderereignisse wie Mausklicks oder gedrückte Tasten an der Tastatur reagieren und Elemente daraufhin dynamisch verändern. Dazu gibt es Event-Handler und auch ein `event`-Objekt. Alle Methoden und Eigenschaften des `Event`-Objekts und alle einzelnen Event-Handler können an dieser Stelle nicht behandelt werden. Blättern Sie das Kapitel 21.2.10 über Event-Handler sowie die Referenz des Objekts `event` (Kapitel 23.18) auf und verschaffen Sie sich einen Überblick über die Möglichkeiten. Hier soll lediglich ein Beispiel die Wirkungsweise verdeutlichen. Ein Beispiel:

```
<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
function SetPos()
{
    document.all.Springer.style.left = window.event.x;
    document.all.Springer.style.top = window.event.y;
}
//-->
</script>
</head><body bgcolor="#FFFFFF" onClick="SetPos()">
<div id="Springer" style="position:absolute; top:100px; left:100px; width:200px;
    height:100px; font-size:16pt; background-color:#000066; color:#FFFFFF00;
padding:1cm;">
Klicken Sie mit der Maus irgendwohin!
</div>
</body>
</html>
```

Das obige Beispiel definiert einen Bereich mit `<div>...</div>`. Verschiedene Stylesheet-Angaben im einleitenden `<div>`-Tag sorgen dafür, dass der Bereich gut sichtbar ist und absolut am Bildschirm positioniert wird. Wenn der Anwender während der Anzeige irgendwo im Anzeigefenster des Browsers mit der Maus hinklickt, wird der definierte `<div>`-Bereich neu positioniert, und zwar so, dass die linke obere Ecke genau an der Stelle beginnt, wo der Anwender angeklickt hat.

Damit dies möglich ist, muss das gesamte Dokument sofort reagieren, wenn der Anwender mit der Maus irgendwohin klickt. Zu diesem Zweck wird im einleitenden `<body>`-Tag der Event-Handler `onClick=` notiert. Beim jedem Mausklick im Anzeigefenster wird dann die Funktion `SetPos()` aufgerufen, die innerhalb des Script-Bereichs definiert ist. Diese Funktion setzt die Werte für die linke obere Ecke des `<div>`-Bereichs neu. Bezug ist dabei der Name (»Springer«), der im `<div>`-Tag mit `id=` vergeben wurde. Mit dem Befehl `document.all.Springer.style`.

left wird die neue Anfangsposition links gesetzt, mit `document.all.Springer.style.top` die neue Anfangsposition oben. Den neuen Wert für links liefert die Eigenschaft `x` des `window.event`-Objekts, den Wert für oben die Eigenschaft `y` dieses Objekts. Der `<div>`-Bereich wird dadurch neu positioniert.

Das Event-Objekt hält ständig alle relevanten Anwendertätigkeiten wie Mausposition, Tastaturereignisse, Mausklicks, gedrückte Maustasten usw. in seinen Objekteigenschaften fest. Durch einfaches Abrufen der Eigenschaften vom Typ `window.event.[Eigenschaft]` können Sie jederzeit aktuelle Ereignisse abfangen. Das Abfangen geschieht durch Notieren von Event-Handlern. Wichtig ist dabei auch, an welcher Stelle Sie ein Ereignis abfangen, also wo Sie den Event-Handler notieren. So könnten Sie im obigen Beispiel den Event-Handler `onClick=` auch im einleitenden `<div>`-Tag notieren. Dann würden Script-Befehle nur dann ausgeführt, wenn der Mausklick innerhalb dieses `<div>`-Bereichs vorkommt.

Ein wichtiges Unterobjekt des Event-Objekts ist das Objekt für das angeklickte Element (`srcElement`). So können Sie etwa mit `window.event.srcElement.tagName` ermitteln, welches HTML-Tag von einem aktuellen Anwenderereignis betroffen ist. In Ihren Scripts können Sie dann entsprechend reagieren.

Es gibt nicht nur den Event-Handler `onClick=`, sondern viele weitere. Wenn Sie beispielsweise im einleitenden `<body>`-Tag die Anweisung `onkeydown="alert(window.event.keyCode)"` notieren, wird bei jedem Drücken einer Taste deren Zeichenwert in dezimaler Form ausgegeben. Denn in der Eigenschaft `keyCode` des `event`-Objekts wird der Tastaturwert festgehalten, sobald der Anwender eine Taste drückt.

**Beachten Sie:** Microsoft stellt zum event-abhängigen Ausführen von Scripts noch eine spezielle Syntax bereit, die Sie ebenfalls verwenden können:

```
<script for="document" event="onkeydown()" language="JScript" type="text/jscript">
  if(event.keyCode == 32) window.location.href="datei2.htm";
</script>
```

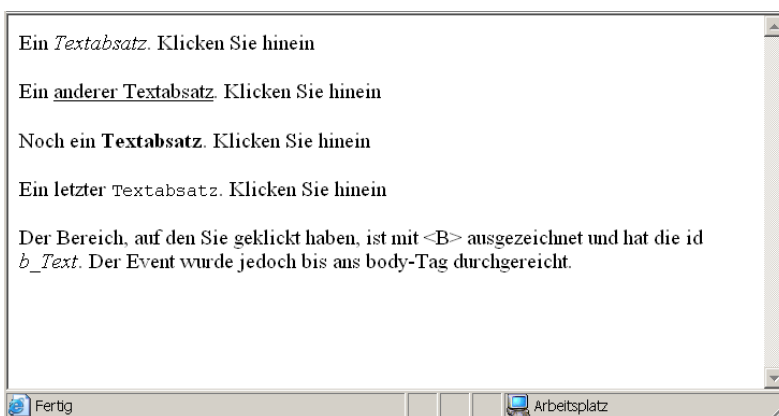
Mit einer solchen Anweisung im einleitenden Script-Tag wird das Script nur dann ausgeführt, wenn das angegebene Ereignis (Event) eintritt. In dem Beispiel wird bewirkt, dass der Browser beim Drücken der Leertaste (sie hat den Tastaturcode 32) zu einer anderen Datei springt.

### 21.3.7 Vererbung bei Events (Event-Bubbling)

Anwenderereignisse (Events) während der Anzeige einer WWW-Seite werden im MS Internet Explorer von »innen nach außen vererbt«. Wenn der Anwender beispielsweise auf einen mit `<b>...</b>` als fett ausgezeichneten Text innerhalb eines mit `<p>...</p>` definierten Textabsatzes klickt, betrifft der Mausklick zunächst nur den fetten Text. Angenommen, der Textabsatz enthält im einleitenden `<p>`-Tag einen Event-Handler `onClick=`, so würde dieser nicht aktiv, da der Mausklick ja nur den als fett markierten Text innerhalb des Absatzes betrifft. Da das `<b>`-Tag keinen Event-Handler enthält, würde der Mausklick unbemerkt bleiben, obwohl er innerhalb des Textabsatzes stattfand, der für den Fall eines Mausklicks einen Event-Handler enthält. Der Bubble-Effekt (wörtlich: *Blaseneffekt*) bewirkt, dass das Ereignis (der Mausklick) zum jeweils nächsthöheren Element im Elementenbaum der HTML-Datei weitergegeben wird. Auf diese Weise gelangt das Ereignis zu dem `<p>`-Tag, das den Event verarbeitet. Würde das `<p>`-Tag auch keinen Event-Handler `onClick=` enthalten, so würde der Bubble-Effekt das Ereignis an das

nächsthöhere Element weitergeben, also z.B. an ein `<div>`-Tag, innerhalb dessen sich der Textabsatz befindet, oder an das `<body>`-Tag. Wenn nirgendwo ein Event-Handler notiert ist, der auf den Mausklick reagiert, passiert einfach nichts. Ein Beispiel:

```
<html>
<head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
function Handle_Click() {
    document.all.Info.innerHTML = "Der Bereich, auf den Sie geklickt
haben, ist mit "<" + window.event.srcElement.tagName + ">
ausgezeichnet und hat die id <i>" +
window.event.srcElement.id + "</i>. Der Event wurde jedoch
bis ans body-Tag durchgereicht.";
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF" onClick="Handle_Click()">
<div id="Inhalt">
<p id="Text_1">Ein <i id="i_Text">Textabsatz</i>. Klicken Sie hinein</p>
<p id="Text_2">Ein <u id="u_Text">anderer Textabsatz</u>. Klicken Sie hinein</p>
<p id="Text_3">Noch ein <b id="b_Text">Textabsatz</b>. Klicken Sie hinein</p>
<p id="Text_4">Ein letzter <tt id="tt_Text">Textabsatz</tt>. Klicken Sie hinein</p>
</div>
<p id="Info">Hier erscheinen Infos zum Klick</p>
</body>
</html>
```



21.9: Das Beispiel zeigt nach einem Mausklick, dass das Event weitergereicht wurde.

Im Beispiel werden diverse Texte in verschiedenen HTML-Auszeichnungen definiert. Jedes HTML-Element erhält dabei auch einen Namen (`id=`). Der unterste Textabsatz mit `id="Info"` dient dazu, dynamisch Informationen darüber auszugeben, welches Element der Anwender

angeklickt hat. Jedes Mal, wenn der Anwender ein Element anklickt, wird die Funktion `Handle_Click()` aufgerufen, die in einem JavaScript-Bereich im Dateikopf definiert ist. Jedes Mal wird in dem Textabsatz »Info« ausgegeben, welches HTML-Tag angeklickt wurde, und welche `id` es hat. Eigentlich betrifft der Mausklick dabei ja nur das jeweilige Element. Keines dieser Elemente hat jedoch einen Event-Handler `onClick=`, der einen Mausklick abfängt und Script-Code ausführt. Dass der Script-Code trotzdem bei jedem Mausklick ausgeführt wird, liegt am Event-Bubbling. Der Mausklick-Event wird immer weiter »nach außen« gegeben und landet schließlich beim `<body>`-Tag, das einen Event-Handler `onClick=` enthält.

Der Bubble-Effekt lässt sich auch ausschalten, falls er nicht erwünscht ist. Dazu müssen Sie in einem JScript/JavaScript-Bereich die Anweisung notieren: `window.event.cancelBubble = true;`

## 21.4 Das ältere DHTML-Modell von Netscape

### 21.4.1 Allgemeines zum DHTML-Modell von Netscape 4.x

Die hier vorgestellte Layer-Technik funktioniert **nur mit Netscape in der Version 4.x**. In der völlig neu programmierten Version 6.0 und den Mozilla-Quellen dazu ist die gesamte Layer-Technik nicht mehr implementiert und funktioniert nicht mehr. Der Grund ist, dass das W3-Konsortium die Layer-Technik abgelehnt hat. Stattdessen wird das Document Object Model (DOM) empfohlen, das auch in Netscape ab Version 6.0 implementiert ist. Die Layer-Technik, wie sie hier beschrieben wird, ist also nur noch im Hinblick auf die Rückwärtskompatibilität zur 4er-Serie der Netscape-Browser von Bedeutung (siehe Kapitel 12).

»Dynamisches HTML« – das ist bei Netscape 4.x **nicht** wie im DOM oder beim älteren Modell des MS Internet Explorer der beliebige nachträgliche Zugriff auf alle Elemente einer HTML-Datei. Sie können mit Netscape 4.x also nicht einfach nachträglich einen Textabsatz, der mit `<p>...</p>` definiert ist, völlig anders formatieren oder in eine Überschrift 1. Ordnung verwandeln.

Dynamisches HTML gibt es bei Netscape 4.x nur im Zusammenhang mit positionierten Elementen. Bei Netscape ist Dynamisches HTML die Möglichkeit, positionierte Layer dynamisch zu behandeln, also ein- und auszublenden, zu verschieben usw.

Die Layer-Technologie von Netscape ist allerdings sehr mächtig. Voraussetzung dabei ist das Definieren von Layern in HTML. Es besteht zwar auch die Möglichkeit, ohne das `layer`-Element Dynamisches HTML für Netscape zu schreiben, doch das ganze Konzept, das Netscape seinem Browser in der Version 4.x als Dynamisches HTML eingebaut hat, basiert auf dem `layer`-Element. Wenn man auf das `layer`-Element verzichten will, bleibt einem nur, die Layer zu »simulieren«.

Die Beispiele auf dieser Seite konzentrieren sich deshalb im Wesentlichen auf die Layer-Technologie. Denn nur so wird der Ansatz deutlich, den Netscape mit seiner Implementierung von Dynamischem HTML verfolgt. Um die Beispiele auf dieser Seite zu verstehen, müssen Sie vor allem die Layer in HTML kennen sowie Vorkenntnisse in JavaScript mitbringen. Für die Layer-Technik ist vor allem das JavaScript-Objekt `layers` entscheidend. Dort werden die einzelnen

Eigenschaften und Methoden systematisch beschrieben, die hier in den Beispielen verwendet werden.

### 21.4.2 Layer-Bereiche dynamisch anzeigen/verstecken

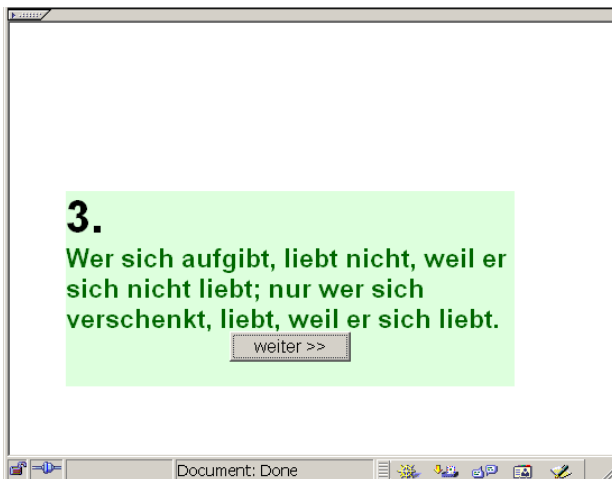
Sie können Layer mit Hilfe von JavaScript dynamisch anzeigen und verstecken. Je nachdem, wie Sie Ihre Layer definieren, können Sie dadurch die verschiedensten Effekte erzielen. Das folgende Beispiel demonstriert, wie Sie eine »Diashow« aus kurzen Texten definieren können. Der Anwender braucht nicht mehr scrollen, sondern klickt sich von einem Text zum nächsten. Die Texte werden in einer »Box« angezeigt. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
var AktuellerText = 0;
var Texte = 3;

function NextText() {
  ++AktuellerText;
  if(AktuellerText >= Texte) AktuellerText = 0;
  for(var i=0;i<Texte;i++)
    document.layers[i].visibility = "hide";
  document.layers[AktuellerText].visibility = "show";
}
//-->
</script>
</head><body>
<layer visibility="show" top="150" left="50" width="400" height="125"
bgcolor="#DDFFDD">
  <p style="color:#006600; font-family:Arial,Helvetica,San-Serif; font-size:14pt;
font-weight:bold;"><span style="color:#000000; font-family:Arial,Helvetica,San-
Serif; font-size:24pt; font-weight:bold;">1.</span><br>
Man kann nur Brücken schlagen zwischen Ufern, die man auseinanderhält.</p>
</layer>
<layer visibility="hide" top="150" left="50" width="400" height="125"
bgcolor="#DDFFDD">
<p style="color:#006600; font-family:Arial,Helvetica,San-Serif; font-size:14pt;
font-weight:bold;"><span style="color:#000000; font-family:Arial,Helvetica,San-
Serif; font-size:24pt; font-weight:bold;">2.</span><br>
Je größer die Erwartung, desto unmöglicher die Erfüllung.</p>
</layer>
<layer visibility="hide" top="150" left="50" width="400" height="125"
bgcolor="#DDFFDD">
<p style="color:#006600; font-family:Arial,Helvetica,San-Serif; font-size:14pt;
font-weight:bold;"><span style="color:#000000; font-family:Arial,Helvetica,San-
Serif; font-size:24pt; font-weight:bold;">3.</span><br>
Wer sich aufgibt, liebt nicht, weil er sich nicht liebt; nur wer
```

```
sich verschenkt, liebt, weil er sich liebt.</p>
</layer>
<layer top="275" left="50" width="400" bgcolor="#DDFFDD">
<center><form name="Formular" action=""><input type="button" value=" weiter >> "
onClick="NextText()"></form></center>
</layer>
</body></html>
```

Im Körper der Beispiel-HTML-Datei werden insgesamt vier Layer definiert. Die ersten drei Layer enthalten Texte, die dynamisch wechseln sollen. Der vierte Layer enthält einen Button. Wenn der Anwender auf den Button klickt, wird eine JavaScript-Funktion mit Namen `NextText()` aufgerufen. Diese Funktion wird im Kopf der HTML-Datei in einem JavaScript-Bereich definiert. Die Funktion sorgt dafür, dass der jeweils nächste Text angezeigt wird.



21.10: Ein Klick auf den Button zeigt nacheinander die verschiedenen Layer an.

Beachten Sie, dass die ersten drei Layer identische Angaben zur Position enthalten. Die Angaben `top="150"` (linke obere Ecke beginnt 150 Pixel von oben), `left="50"` (linke obere Ecke beginnt 50 Pixel von links), `width="400"` (Breite des Layers: 400 Pixel), und `height="125"` (Höhe des Layers: 125 Pixel), sind in allen Layern die gleichen. Die drei Layer unterscheiden sich lediglich durch die Angabe von `visibility`. Der erste Layer wird mit `visibility="show"` auf »sichtbar« gesetzt, die beiden anderen mit `visibility="hide"` auf »unsichtbar«. Dadurch wird erreicht, dass zunächst nur der erste Layer am Bildschirm angezeigt wird.

Der vierte Layer, also der mit dem Kontroll-Button, enthält ebenfalls Angaben zur Position. Er hat die gleiche Breite wie die beiden ersten Layer und beginnt auch links an der gleichen Position. In der Höhe beginnt er genau 125 Pixel tiefer als die beiden ersten Layer. Da diese eine Höhe von 125 Pixeln haben, schließt der »Kontroll-Button-Layer« also nahtlos an die oberen Layer an. Er enthält keine Angabe zur Sichtbarkeit. Das bedeutet, er ist sichtbar, und das wird sich auch im Verlauf des JavaScripts nicht ändern. Denn der Inhalt dieses Layers steuert ja das Geschehen.

In dem JavaScript-Bereich im Kopf der Datei werden zunächst zwei globale Variablen definiert. Da sie außerhalb jeder Funktion stehen, behält ihr aktueller Wert so lange Gültigkeit, wie die HTML-Datei angezeigt wird. In der Variablen `AktuellerText` wird die Indexnummer des aktuellen Textlayers gespeichert. In `Texte` wird die Anzahl der dynamisch austauschbaren Texte bzw. Layer gespeichert. Sie könnten anstelle der drei Layer mit Texten auch 10 oder 100 solcher Layer definieren. Entsprechend der Anzahl der definierten Layer müssten Sie nur den Anfangswert von Texten anders zuweisen.

Innerhalb der Funktion `NextText()` wird zunächst die Nummer von `AktuellerText` erhöht, denn es soll ja der jeweils nächste Text angezeigt werden. Anschließend werden in einer `for`-Schleife alle Layer mit Texten auf »unsichtbar« gesetzt. Am Ende wird der Layer desjenigen Textes, der angezeigt werden soll (`AktuellerText`) auf »sichtbar« gesetzt. Dadurch entsteht der Diashow-Effekt.

**Beachten Sie:** Netscape dehnt die Layer entsprechend ihres Inhalts aus. Wenn der Inhalt eines Textes im obigen Beispiel größer ist als der definierte Layer-Bereich, überschneidet er sich mit dem Layer des Kontroll-Buttons, jedoch so, dass der Layer mit dem Kontroll-Button über dem unteren Teil des Layers mit dem Text liegt. Der Grund dafür ist, dass der Layer mit dem Kontroll-Button hinter dem Layer mit dem Text definiert wird und daher im »Stack« (Stapel) weiter oben liegt. Wenn Sie die Größe eines Layers unabhängig vom Inhalt fest machen wollen, müssen Sie das Attribut `clip=` benutzen (siehe auch Kapitel 12.1.4) *Anzeigebereich von Layern beschneiden*).

### 21.4.3 Layer-Bereiche dynamisch überlappen

Sie können Layer so positionieren, dass sie sich überlappen. Das macht ohne JavaScript meistens allerdings wenig Sinn, denn der Anwender soll die Inhalte der Layer ja sehen können. Mit Hilfe von JavaScript können Sie jedoch bewirken, dass der Anwender sich einen einzelnen Layer in den Vordergrund holen kann, um dessen Inhalt anzuzeigen. Auf diese Weise können Sie auf einer WWW-Seite durchaus verschiedene Inhalte unterbringen, für die normalerweise mehrere Seiten angebracht wären. Im folgenden Beispiel sind die Layer wie in einem Karteikastenstapel aufgereiht. Der Anwender kann einen Layer »herausgreifen«. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
layer.Text { padding:5px; font-size:9pt; font-family:Arial,Helvetica,Sans-Serif; }
layer.Kopf { font-size:9pt; font-family:Arial,Helvetica,Sans-Serif; font-
weight:bold; text-align:right; }
a:link, a:visited, a:active { color:white; }
a.normal:link { color:#AA5522; text-decoration:underline; }
a.normal:visited { color:#772200; text-decoration:underline; }
a.normal:active { color:#000000; text-decoration:underline; }
-->
</style>
<script type="text/javascript">
<!--
var aktiv = "Home";
```

```

function Hervorheben(neu_aktiv)
{
    document.layers['Feedback'].zIndex = 1;
    document.layers['Support'].zIndex = 2;
    document.layers['Produkte'].zIndex = 3;
    document.layers['Home'].zIndex = 4;
    document.layers[aktiv].bgColor = "#CCCCCC";
    document.layers[aktiv].layers[0].bgColor = "#666666";
    document.layers[aktiv].layers[1].visibility = "hide";
    document.layers[neu_aktiv].zIndex = 5;
    document.layers[neu_aktiv].bgColor = "#FFCCFF";
    document.layers[neu_aktiv].layers[0].bgColor = "#000000";
    document.layers[neu_aktiv].layers[1].visibility = "show";
    aktiv = neu_aktiv;
}
//-->
</script>
</head><body>
<layer id="Feedback" top="130" left="10" width="360" height="230"
bgcolor="#CCCCCC">
    <layer class="Kopf" width="360" height="25" bgcolor="#666666">
        <a href="javascript:Hervorheben('Feedback')">Feedback</a>
    </layer>
    <layer class="Text" top="25" visibility="hide">
        Hier Ihr Feedback:
        <form name="Formular" action=""><textarea rows="5" cols="25">Bla
Bla</textarea></form>
    </layer>
</layer>
<layer id="Support" top="155" left="70" width="360" height="230" bgcolor="#CCCCCC">
    <layer class="Kopf" width="360" height="25" bgcolor="#666666">
        <a href="javascript:Hervorheben('Support')">Support</a>
    </layer>
    <layer class="Text" top="25" visibility="hide">
        Unser Suport ist wirklich Bla Bla
    </layer>
</layer>
<layer id="Produkte" top="180" left="130" width="360" height="230"
bgcolor="#CCCCCC">
    <layer class="Kopf" width="360" height="25" bgcolor="#666666">
        <a href="javascript:Hervorheben('Produkte')">Produkte</a>
    </layer>
    <layer class="Text" top=25 visibility=hide>
        Unsere Produkte sind ja so wahnsinnig Bla Bla
    </layer>
</layer>
<layer id="Home" top="205" left="190" width="360" height="230" bgcolor="#FFCCFF">

```

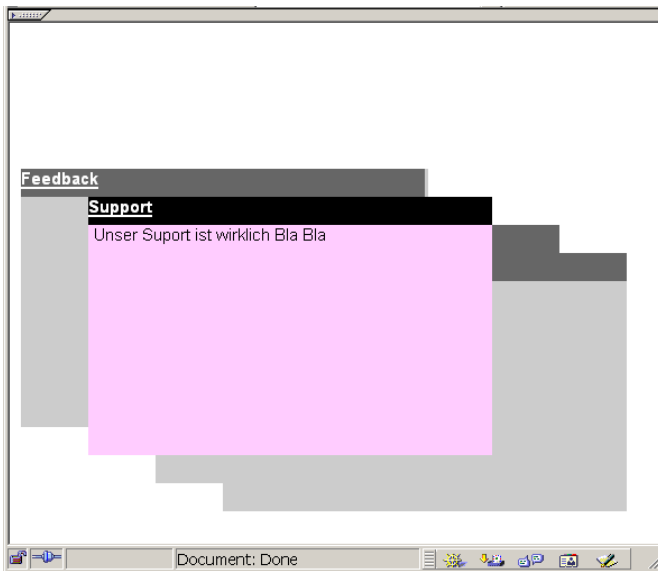


```

<layer class="Kopf" width="360" height="25" bgcolor="#000000">
  <a href="javascript:Hervorheben('Home')">HOME</a>
</layer>
<layer class="Text" top="25" visibility="show">
  Unser Home ist unser Bla Bla
</layer>
</layer>
</body></html>

```

Im Beispiel werden vier »große« Layer definiert: einer für »Feedback«, einer für »Support«, einer für »Produkte« und einer für »Home«, was bedeuten soll, dass dieser Layer eine Art Homepage, einen Home-Layer darstellt. Dieser Layer wird auch zuerst im Vordergrund angezeigt. Und zwar deshalb, weil er als letzter definiert wird. Wenn Sie bei Layern kein Attribut `z-index=` notieren, liegt immer derjenige Layer ganz oben, der als Letzter definiert wird.



21.11: Ein Mausklick auf den Link holt den jeweiligen Layer nach vorne.

Dem Beispiel können Sie auch entnehmen, dass innerhalb der vier großen Layer jeweils zwei »innere« Layer definiert werden. Layer, die innerhalb anderer Layer definiert werden, werden relativ zu ihrem Eltern-Layer positioniert. Dazu müssen Sie sich einen Layer vorstellen wie einen neuen kleinen Dateikörper. Alles, was innerhalb davon notiert wird, wird relativ dazu positioniert.

Die Angaben zu `top=`, `left=`, `height=` und `width=` in den Layern im obigen Beispiel sind alle genau ermittelt, und zwar so, dass die Layer optisch einen sauber sortierten Karteikastenstapel darstellen. Versuchen Sie, die Abhängigkeiten der Positionierungsangaben im Beispiel zu ermitteln. So können Sie erkennen, wie Sie vorgehen müssen, um mehrere Layer nach einem von Ihnen gewünschten Kriterium ordentlich verteilt im Anzeigefenster zu platzieren.

Jeder der »großen« Layer enthält im obigen Beispiel zwei innere Layer: einen Layer, der wie eine Titelleiste wirkt und einen Verweis enthält, und einen Layer, der den eigentlichen Inhalt enthält. Die Verweise in den Titelleisten-Layern rufen eine JavaScript-Funktion namens `Hervorheben()` auf. Diese Funktion bewirkt das dynamische Hervorholen des »großen« Layers, der zu dem angeklickten Verweis gehört (also des Eltern-Layers von dem Layer, in dem der Verweis steht).

In dem Script-Bereich, in dem die Funktion `Hervorheben()` steht, wird vor der Funktion eine Variable `aktuell` definiert. Sie bekommt zu Beginn den Namen des vorne liegenden Layers zugewiesen. Das ist derjenige Name, der beim entsprechenden Layer im Attribut `id=` vergeben wurde. Die Funktion `Hervorheben()` erwartet einen Parameter, nämlich den Namen desjenigen Layers, der anstelle des bisher vorne liegenden Layers hervorgeholt werden soll. Bei den Verweisen, die die Funktion aufrufen, wird denn auch wieder der Name übergeben, der im entsprechenden Layer bei `id=` vergeben ist.

Innerhalb der Funktion `Hervorheben()` wird zunächst die normale Layer-Reihenfolge hergestellt. Das geschieht durch Zuweisen einer Zahl an die Layer-Objekteigenschaft `zIndex`. Je höher die Zahl, desto weiter vorne im Stapel liegt der Layer. Das Wiederherstellen der normalen Layer-Reihenfolge bewirkt, dass der »Karteikastenstapel«, den das Beispiel darstellt, immer aufgeräumt erscheint.

Anschließend wird der bisher aktive Layer, bezeichnet durch `document.layers[aktiv]`, mit mehreren Anweisungen behandelt. Die Angabe `aktiv` in den eckigen Klammern zeigt zunächst auf den Layer "Home". Am Ende der Funktion wird der Wert der Variablen `aktiv` jedoch geändert. Dort wird ihr der Name des neuen hervorgehobenen Layers zugewiesen. Dadurch ist beim nächsten Aufruf der Funktion dieser Layer derjenige, der von `document.layers[aktiv]` betroffen ist. Die Anweisungen, die den bisher aktiven Layer betreffen, weisen diesem Attribut zu, die ihn optisch zu einem »unten liegenden« Layer im Karteikastenstapel machen.

Derjenige Layer mit dem Namen, der mit dem Parameter `neu_aktiv` übergeben wurde, erhält dagegen diejenigen optischen Attribute, die ihn als hervorgehobenen Layer auszeichnen. Er bekommt auch die höchste Zahl für `zIndex`. Dadurch wird er als oberster Layer angezeigt.

**Beachten Sie:** Im Layer `Feedback` ist ein Formular definiert. Dieses Formular wird im Netscape 4 mit `document.layers.Feedback.document.Formular` angesprochen, da es Bestandteil des Layers `Feedback` ist. Ein Layer stellt aus Sicht von Netscape ein Dokument im Dokument dar und hat ein eigenes `document`-Objekt.

#### 21.4.4 Layer-Bereiche dynamisch bewegen

Sie können Layer mit beliebigen Inhalten über das Anzeigefenster springen oder wandern lassen. Das Springen zu einer Stelle ist dabei nur ein einziger Befehl, da dies als Methode im Layer-Objekt von JavaScript zur Verfügung steht. Wenn Sie dagegen zeitgesteuerte Bewegungseffekte wollen, müssen Sie die Positionsrechnung vom Startpunkt bis zum Endpunkt des Bewegungsvorgangs »zu Fuß« programmieren. Das folgende Beispiel zeigt, wie so etwas im Prinzip funktioniert.

```
<html><head><title>Test</title>
<style type="text/css">
<!--
p.Nav { font-size:9pt; font-weight:bold }
```

```

a.Navlink:link { color:#FF0000 }
a.Navlink:visited { color:#FF0000 }
a.Navlink:active { color:#000000 }
-->
</style>
<script type="text/javascript">
<!--
function hBeweg(LayerNum,bis_links,Pixel,Pause) {
  var Restpixel = bis_links - document.layers[LayerNum].left;
  if(bis_links < document.layers[LayerNum].left) Pixel=0-Pixel;
  if(Math.abs(Restpixel) > Math.abs(Pixel))
    document.layers[LayerNum].moveBy(Pixel,0);
  else
    document.layers[LayerNum].moveBy(Restpixel,0);
  if(document.layers[LayerNum].left != bis_links)
window.setTimeout('hBeweg('+LayerNum+', '+bis_links+', '+Math.abs(Pixel)+')', '+Pause+'
');
}

function vBeweg(LayerNum,bis_oben,Pixel,Pause) {
  var Restpixel = bis_oben - document.layers[LayerNum].top;
  if(bis_oben < document.layers[LayerNum].top) Pixel=0-Pixel;
  if(Math.abs(Restpixel) > Math.abs(Pixel))
    document.layers[LayerNum].moveBy(0,Pixel);
  else
    document.layers[LayerNum].moveBy(0,Restpixel);
  if(document.layers[LayerNum].top != bis_oben)
window.setTimeout('vBeweg('+LayerNum+', '+bis_oben+', '+Math.abs(Pixel)+')', '+Pause+'
);
}
//-->
</script>
</head><body>
<layer left="10" top="10" width="450" height="250" bgcolor="#FFFF66">
<h1>Einen springen lassen...</h1>
<p>Springe nach:
<a class="Navlink" href="javascript:document.layers[0].moveTo(10,10)">links
oben</a> |
<a class="Navlink" href="javascript:document.layers[0].moveTo(260,10)">rechts
oben</a> |
<a class="Navlink" href="javascript:document.layers[0].moveTo(10,260)">links
unten</a> |
<a class="Navlink" href="javascript:document.layers[0].moveTo(260,260)">rechts
unten</a><br></p>
<p>Bewege nach:
<a class="Navlink" href="javascript:hBeweg(0,10,4,2)">links</a> |
<a class="Navlink" href="javascript:hBeweg(0,260,4,2)">rechts</a> |

```

```

<a class="Navlink" href="javascript:vBeweg(0,10,4,2)">oben</a> |
<a class="Navlink" href="javascript:vBeweg(0,260,4,2)">unten</a></p>
</layer>
</body></html>

```



21.12: Das Layer springt per Script im Browser-Fenster hin und her.

Innerhalb des Dateikörpers der HTML-Datei im obigen Beispiel wird ein Layer definiert. Dieser Layer enthält insgesamt acht Verweise. Bei jedem Verweis wird JavaScript-Code ausgeführt. Dabei wird der Befehl, den der Verweis dem Anwender anbietet, in die Tat umgesetzt. Alle Verweise sorgen für eine Neupositionierung des Layers, in dem sie sich selber befinden. Das muss nicht so sein – es ist im obigen Beispiel einfach zu Demo-Zwecken so gelöst. Der Layer, den es dynamisch zu positionieren gilt, könnte auch etwas ganz anderes enthalten, etwa Grafiken, Tabellen usw.

Die ersten vier Verweise führen lediglich Sprünge aus, d.h. beim Klicken auf einen dieser Verweise wird der Layerinhalt einfach sofort an einer anderen Stelle im Anzeigefenster angezeigt. Für diese Aufgabe stellt das Layer-Objekt von JavaScript die Methode `moveTo()` zur Verfügung. Mit einer Anweisung wie `document.layers[0].moveTo(...)` positionieren Sie den ersten Layer der Datei (der den Index 0 hat) an einer neuen Position. Die Methode `moveTo()` setzt die neue Position der linken oberen Ecke des Layers. Dazu erwartet die Methode zwei Parameter, und zwar zuerst den Wert für links, und dann den Wert für oben. Mit einer Anweisung wie `document.layers[0].moveTo(260,10)` wird also die linke obere Ecke des Layers auf 260 Pixel von links und 10 Pixel von oben gesetzt. Die Angaben beziehen sich auf das jeweilige Elternelement des Layers. Da der Layer im obigen Beispiel kein Layer innerhalb eines anderen Layers ist, ist das normale Anzeigefenster das Elternelement.

Während das einfache Neupositionieren mit einer einzigen JavaScript-Anweisung möglich ist, die im obigen Beispiel direkt in den entsprechenden Verweisen steht, muss zum dynamischen Bewegen von Layern eine eigene Funktion aufgerufen werden, die den Bewegungsablauf be-

stimmt. Im obigen Beispiel gibt es zwei solcher Funktionen. Die Funktion `hBeweg()` kann Layer horizontal nach links oder rechts verschieben, die Funktion `vBeweg()` leistet das Gleiche, nur vertikal nach oben oder unten. Entsprechend werden die Funktionen aufgerufen. Wenn der Anwender einen der Verweise zum Bewegen nach rechts oder links angeklickt, wird `hBeweg()` aufgerufen, und bei den Verweisen zum Bewegen nach unten oder oben wird `vBeweg()` aufgerufen.

Beide Funktionen sind allgemein geschrieben. Deshalb erwarten sie als Input auch verschiedene Parameter, die beim Funktionsaufruf zu übergeben sind:

- `LayerNum` ist die Indexnummer des Layers, der bewegt werden soll. Da im Beispiel immer der gleiche und einzige vorhandene Layer bewegt werden soll, wird beim Aufruf der Funktionen immer dessen Indexnummer 0 übergeben.
- `bis_links` (bei `hBeweg()`) bzw. `bis_oben` (bei `vBeweg()`) ist der neue Endwert, bei dem die Bewegung gestoppt werden soll. Je nachdem, ob der Wert, den Sie dabei übergeben, größer oder kleiner ist als die aktuelle Position des Layers, ergibt sich automatisch die Bewegungsrichtung.
- `Pixel` ist die Anzahl der Pixel, um die pro Bewegungsschritt weiter gesprungen wird. Je höher der Wert ist, den Sie dafür übergeben, desto schneller und ruckartiger die Bewegung. Im obigen Beispiel wird in allen Fällen 4 übergeben.
- `Pause` ist ein weiterer Parameter zur Beeinflussung der Geschwindigkeit. Je höher der Wert, desto langsamer die Bewegung.

Ein Bewegen von Layern besteht in nichts anderem als einem schrittweise neuen Positionieren des Layers. In den Funktionen `hBeweg()` und `vBeweg()` stehen jeweils etwa in der Mitte zwei Anweisungen der Art `document.layers[LayerNum].moveBy(...)`, von denen immer nur eine ausgeführt wird. Das ist diejenige Anweisung, mit der der Layer jeweils neu positioniert wird. Es handelt sich einfach um den Aufruf der Methode `moveBy()` des Layer-Objekts. Diese Methode funktioniert genauso wie `moveTo()`, nur mit relativen Angaben statt mit absoluten.

Damit der Layer nicht nur einmal, sondern weiter bewegt wird, rufen sich die beiden Funktionen `hBeweg()` und `vBeweg()` am Ende jeweils selbst wieder auf. Der Schleifen-Effekt, der dadurch entsteht, wird erst abgebrochen, wenn der gewünschte Endwert erreicht ist. Der Selbstaufruf der Funktion ist in die JavaScript-Methode `window.setTimeout()` eingebunden. Dadurch kann die Verzögerung berücksichtigt werden, die mit dem Parameter `Pause` übergeben wurde. Der Selbstaufruf der Funktion wirkt etwas kompliziert. Das liegt daran, dass die Funktion sich auch wieder mit den von ihr erwarteten Parametern aufrufen muss. Leider ist es nicht möglich, dazu einen normalen Funktionsaufruf zu notieren und einfach die übergeben bekommenen Parameter wieder zu übergeben. Stattdessen wird eine Zeichenkette zusammengebaut, die die Funktion beim Neuaufruf mit tatsächlichen Werten versorgt.

Die übrigen Anweisungen in den beiden Funktionen dienen zur Kontrolle des Bewegungsablaufs, vor allem dazu, dass trotz unterschiedlicher Aufrufparameter am Ende tatsächlich die gewünschte Endposition erreicht wird. So ist `Restpixel` eine Hilfsvariable, die ermittelt, wie viele Pixel noch zwischen aktueller Layerposition und dem gewünschten Endwert liegen. Wichtig ist auch die korrekte Bestimmung der Bewegungsrichtung. Wenn der gewünschte Endwert der Bewegung kleiner ist als die Position des Layers beim Start der Bewegung, dann muss der Schrittwert, der mit dem Parameter `Pixel` übergeben wird, als negative Zahl interpretiert wer-

den. Denn nur so kann es »rückwärts« gehen, also nach links oder nach oben. Mit dem Ausdruck `Pixel=0-Pixel` wird der Schrittwert bei Bedarf auf eine negative Zahl »umgepolt«.

**Beachten Sie:** Komplexere Bewegungsabläufe benötigen entsprechend komplexere Programmierung. Bei diagonalen Bewegungen ist es nur einfach, solange die Anzahl der Pixel, um die pro Bewegungsschritt weiter positioniert wird, horizontal und vertikal immer proportional ist. Sobald Sie aber »krumme« Aufgaben lösen wollen, etwa die, von oben 10 links 10 nach oben 86 links 623 auf möglichst geradem Weg zu kommen, dürfen Sie in die Tiefen der Arithmetik eintauchen. Und wenn Sie gar Kreis- oder Ellipsenbewegungen wollen, müssen Sie entsprechende geometrische Formeln kennen und die dafür vorgesehenen Methoden des `Math`-Objekts von JavaScript benutzen.

### 21.4.5 Layer-Bereiche dynamisch vergrößern/verkleinern

Durch dynamisches Verändern des angezeigten Bereichs eines Layers lassen sich unzählige interessante Effekte erzielen, zum Beispiel zum dynamischen Aufklappen und Zuklappen von Teilen einer WWW-Seite. Das folgende Beispiel bewirkt drei »Blöcke«, von denen zunächst nur Titelzeilen sichtbar sind. Jede Titelzeile enthält einen Verweis. Beim Anklicken des Verweises wird unterhalb der Titelzeile ein Bereich mit Inhalt nach unten aufgerollt. Beim nächsten Anklicken des Verweises wird er wieder zugerollt. Ein Beispiel:

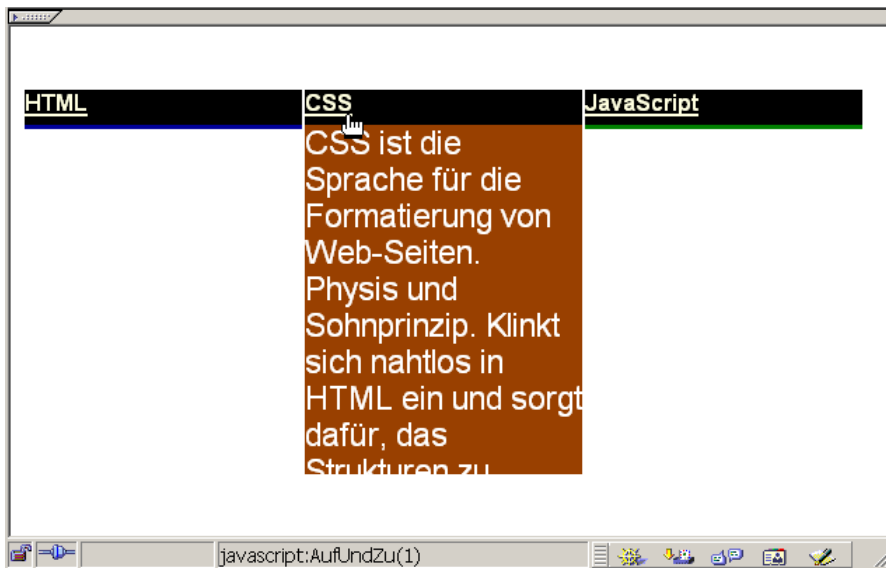
```
<html><head><title>Test</title>
<style type="text/css">
<!--
layer.Text { padding:5px; font-size:14pt; font-family:Arial,Helvetica,Sans-Serif;
color:#FFFFFF }
layer.Kopf { padding:5px; font-size:10pt; font-family:Arial,Helvetica,Sans-Serif;
font-weight:bold; }
a:link, a:visited, a:active { color:#FFFFE0; }
a.normal:link { color:#AA5522; text-decoration:underline; }
a.normal:visited { color:#772200; text-decoration:underline; }
a.normal:active { color:#000000; text-decoration:underline; }
-->
</style>
<script type="text/javascript">
<!--
var Status = new Array("zu","zu","zu");
function AufUndZu(LayerNum) {
  var i;
  if(Status[LayerNum] == "zu") {
    for(i=2;i<250;++i)
      document.layers[LayerNum].clip.bottom = i;
    Status[LayerNum] = "auf";
  }
  else {
    for(i=250;i>2;--i)
      document.layers[LayerNum].clip.bottom = i;
    Status[LayerNum] = "zu";
  }
}
```

```

    }
}
//-->
</script>
</head><body>
<layer class="Text" left="10" top="70" width="198" height="250" bgcolor="#000099"
clip="0,0,198,2">
HTML ist die Sprache für die Strukturierung von Hypertext-Projekten. Logos und
Godfather aller
Web-Seiten. Vermutlich das erfolgreichste Dateiformat der EDV-Geschichte.
</layer>
<layer class="Text" left="210" top="70" width="198" height="250" bgcolor="#994000"
clip="0,0,198,2">
CSS ist die Sprache für die Formatierung von Web-Seiten. Physis und Sohnprinzip.
Klinkt sich
nahtlos in HTML ein und sorgt dafür, das Strukturen zu Farben und Formen werden.
</layer>
<layer class="Text" left="410" top="70" width="198" height="250" bgcolor="#008000"
clip="0,0,198,2">
JavaScript ist die Sprache für Dynamische Web-Seiten. Kinesis und Geistprinzip.
Klinkt sich nahtlos
in HTML ein und zunehmend auch in CSS. Ist allerdings am schwersten zu verstehen.
</layer>
<layer class="Kopf" left="10" top="45" width="198" height="25" bgcolor="#000000">
<a href="javascript:AufUndZu(0)">HTML</a>
</layer>
<layer class="Kopf" left="210" top="45" width="198" height="25" bgcolor="#000000">
<a href="javascript:AufUndZu(1)">CSS</a>
</layer>
<layer class="Kopf" left="410" top="45" width="198" height="25" bgcolor="#000000">
<a href="javascript:AufUndZu(2)">JavaScript</a>
</layer>
</body></html>

```

In dem Beispiel werden insgesamt sechs Layer definiert. Die ersten drei Layer sind diejenigen, die auf- und zuklappbar sind. Zunächst sind diese Layer zugeklappt. Das wird durch das Attribut `clip=` erreicht. Durch diese Angabe lässt sich der angezeigte Bereich eines Layers beliebig einschränken. Die auf-/zuklappbaren Layer im obigen Beispiel etwa haben durch die Angaben `width="198" height="250"` eine Breite von 198 und eine Höhe von 250 Pixeln. Durch die Angabe `clip="0,0,198,2"` wird Folgendes erreicht: Trotz der definierten Breite und Höhe ist nur der Bereich von links 0 oben 0 bis links 198 oben 2 sichtbar. Das ist nur ein ganz dünner Streifen, der aus optischen Gründen zugelassen wird. Mit `0,0,198,0` wäre von den Layern überhaupt nichts zu sehen.



21.13: Ein Klick auf den Link »klappt« das Layer auf und zeigt den Text.

Die drei unteren Layer sind mit Hilfe der Angaben zu `top=`, `left=`, `width=` und `height=` so positioniert, dass sie genau über den auf-/zuklappbaren Layern stehen. Diese »Kopf«-Layer enthalten Verweise. Bei jedem Klick auf einen Verweis in einem der Kopf-Layer wird der Layer unterhalb davon auf- oder wieder zugeklappt. Dazu wird bei jedem Klick auf einen der Verweise eine JavaScript-Funktion aufgerufen: die Funktion `AufUndZu()`, die im Kopf der Datei in einem JavaScript-Bereich definiert ist.

In dem JavaScript-Bereich wird vor der Funktion `AufUndZu()` bereits ein Array, bestehend aus drei Werten, definiert: der Array `Status`. Alle seine drei Werte erhalten den Anfangszustand "zu". Da dieser Array außerhalb jeder Funktion definiert ist, stehen seine Daten jederzeit zur Verfügung.

Innerhalb der Funktion `AufUndZu()` wird geprüft, ob ein Layer derzeit auf- oder zugeklappt ist. Dazu bekommt die Funktion beim Aufruf einen Parameter `LayNum` übergeben. Das ist die Indexnummer des Layers, der auf- oder zugeklappt werden soll. Da im Beispiel die ersten drei definierten Layer davon betroffen sein sollen, kommen also nur die Indexnummern 0, 1 und 2 in Frage. Beim Aufruf der Funktion in den unteren drei Layern werden denn auch genau diese drei Werte übergeben. Mit der Abfrage `if(Status[LayNum] == "zu")` vergleicht die Funktion `AufUndZu()`, was im Array `Status` als Wert für die übergebene Indexnummer steht. Wenn dort "zu" steht, wird der Layer aufgeklappt. Wenn er aufgeklappt wird, wird der Wert im Array am Ende auf den Wert "auf" gesetzt. Dadurch gelangt die Funktion bei der nächsten Abfrage für den gleichen Layer in den `else`-Zweig, wo sie genau die umgekehrten Arbeiten durchführt.

Das eigentliche Auf- und Zuklappen geschieht innerhalb der `for`-Schleife. Dort wird beim Aufklappen von 2 bis 250 gezählt und beim Zuklappen von 250 zurück auf 2. Abhängig vom aktuellen Schleifenwert wird die Unterseite des Clip-Bereichs des Layers mit `document.layers[LayNum].clip.bottom=` dynamisch verändert. Die Grenzwerte von 2 und 250 hängen mit den definierten Layern zusammen: Die 2 ist jener Wert, der bei den Angaben `clip="0,0,198,2"`



vorkommt, und 250 ist die Höhe der definierten Layer (`height="250"`). Beim Aufklappen wird also der Clip-Bereich im Script dynamisch auf 0,0,198,250 verändert. Das bedeutet: Der Layer ist vollständig sichtbar. Beim Zuklappen wird dagegen der Clip-Anfangszustand von 0,0,198,2 wieder hergestellt.

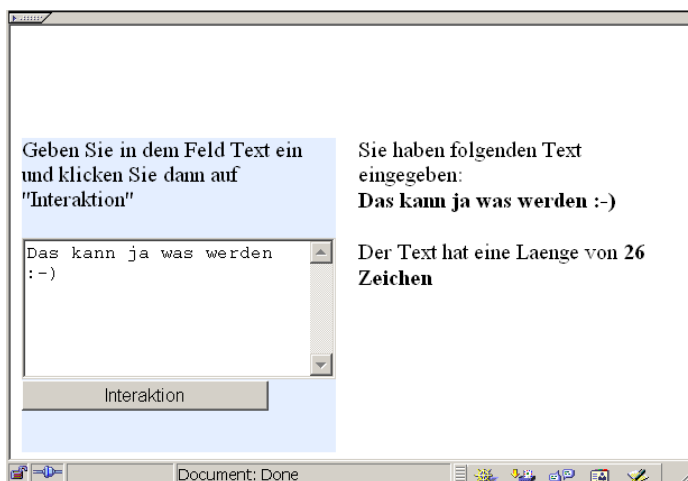
#### 21.4.6 Layer-Inhalte dynamisch austauschen

Einen der großen Vorteile des Microsoft-Ansatzes für Dynamisches HTML, nämlich das beliebige Austauschenkönnen von HTML-Elementinhalten, können Sie mit Hilfe von Netscape-Layern zumindest »nachstellen«. Der Grund ist, dass Netscape einen Layer wie ein eigenes kleines neues Dokument innerhalb des Elterndokuments behandelt. So sind auch alle JavaScript-Objekte darauf anwendbar, die unterhalb des `document`-Objekts liegen. Das folgende Beispiel verdeutlicht, wie dies bei Layern im Einzelnen funktioniert, und welche Möglichkeiten sich dabei eröffnen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function DatenSchreiben() {
    var Text = document.Input.document.Eingabe.Feld.value;
    var Laenge = document.Input.document.Eingabe.Feld.value.length;
    document.Output.document.clear();
    document.Output.document.write("<p>Sie haben folgenden Text eingegeben:<br>");
    document.Output.document.write("<b>" + Text + "</b></p>");
    document.Output.document.write("<p>Der Text hat eine Laenge von ");
    document.Output.document.write("<b>" + Laenge + " Zeichen</b></p>");
    document.Output.document.close();
}
//-->
</script>
</head><body>
<layer id="Input" bgcolor="#E4EEFF" style="padding:5mm" left="10" top="100"
width="280" height="280">
<p>Geben Sie in dem Feld Text ein und klicken Sie dann auf "Interaktion"</p>
<form name="Eingabe" action="">
<textarea name="Feld" cols="25" rows="6" wrap="virtual"></textarea><br>
<input type="button" value="Interaktion" width="220" onClick="DatenSchreiben()">
</form>
</layer>
<layer id="Output" style="padding:5mm" left="310"
top="100" width="280" height="280">
</layer>
</body></html>
```

Im Beispiel werden zwei Layer definiert: einer mit dem Namen (`id`) `Input` und einer mit dem Namen `Output`. Beide Layer sind so positioniert, dass der Layer für den Output neben dem Layer für den Input erscheint. Der Layer für den Output erhält jedoch zunächst keinen Inhalt.

Im Layer für den Input wird dagegen ein Formular definiert. Das Formular besteht aus einem mehrzeiligen Eingabefeld und einem Button. Wenn der Anwender auf den Button klickt, wird die JavaScript-Funktion `SchreibeDaten()` aufgerufen, die im Kopf der Datei innerhalb eines JavaScript-Bereichs definiert ist.



21.14: Das Script beschreibt den rechten Bereich dynamisch, während die Seite bereits angezeigt wird.

Die Funktion ermittelt zunächst den eingegebenen Text und dessen Länge. Anschließend schreibt sie diese Daten zur Kontrolle in den Layer mit dem Namen `Output`. Wichtig ist, dass vor dem dynamischen Schreiben des Inhalts die Methode `clear()` aufgerufen wird, die den aktuellen Inhalt des Layers löscht, und nach dem Schreiben die Methode `close()`, die den Vorgang abschließt, sodass beim nächsten Mal ein neuer Schreibzugriff möglich ist. Dazwischen werden die eigentlichen Daten geschrieben, und zwar mit der Methode `write()`.

Alle drei Methoden sind altbekannte Methoden des `document`-Objekts von JavaScript. Das besondere im Fall der Layer ist, dass das `document`-Objekt bei den Anweisungen gleich doppelt verwendet wird. Ein Beispiel ist die Anweisung: `document.Output.document.clear();`. Auch zu Beginn der Funktion, beim Ermitteln des Formularfeldinhalts, kommt diese doppelte Verwendung von `document` vor. Solche Konstruktionen müssen Sie so verstehen: Über das vorne notierte `document` sind Layer-Objekte ansprechbar, da diese unterhalb des `document`-Objekts liegen. `Output` ist im Beispiel der Name eines Layers. Dadurch wird dieser Layer angesprochen. Ein Layer gilt nun aber selbst wieder als »Dokument«. Deshalb müssen Sie dahinter nochmals `document` und dahinter erlaubte Eigenschaften und Methoden des `document`-Objekts notieren. Die Anweisungen beziehen sich dann auf die Elemente innerhalb des entsprechenden Layers. Im obigen Beispiel etwa kommen Sie an den Formularfeldinhalt nur mit `document.Input.document.Eingabe.Feld.value` (`document.Eingabe.Feld.value` würde dagegen eine Fehlermeldung erzeugen).

**Beachten Sie:** Der Phantasie sind hierbei kaum Grenzen gesetzt. Alles, was Sie beispielsweise durch Interaktion mit dem Anwender mit Hilfe eines Formulars in JavaScript ermitteln und

berechnen können, können Sie auf diese Weise dynamisch ausgeben. Mit Methoden wie `window.setTimeout()` können Sie auch zeitgesteuert Inhalte in Layer schreiben.

### 21.4.7 Layer ohne layer-Element simulieren

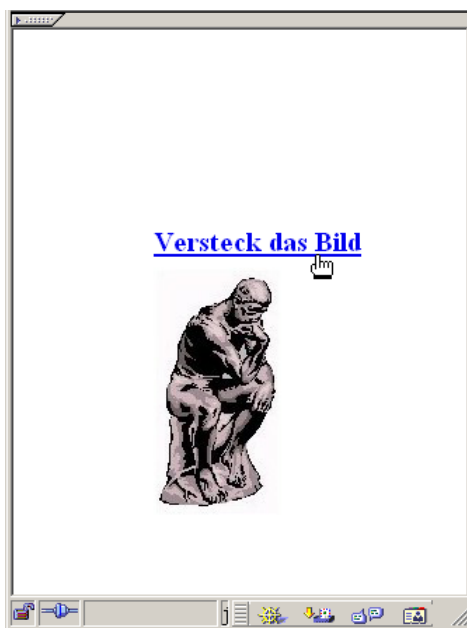
Unter dem Druck von Microsoft und angesichts der Tatsache, dass das W3-Konsortium keine Anstalten machte, das `layer`-Element und damit die Grundlage für Dynamisches HTML nach dem Netscape-Ansatz in den HTML-Standard zu übernehmen, hat Netscape die Fähigkeit zum dynamischen Positionieren auch auf die beiden »abstrakten« HTML-Elemente `<div>...</div>` und `<span>...</span>` ausgedehnt. Wenn Sie innerhalb solcher HTML-Tags mit Hilfe von Stylesheet-Angaben zur Positionierung des Elements notieren, können Sie mit JavaScript auf solche Angaben zugreifen. Netscape interpretiert die Angaben dann genau so wie bei Layern. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript">
<!--
function Zeigen() {
  document.BildAnzeige.visibility = "visible";
  document.ZeigenVerweis.visibility = "hidden";
  document.VersteckenVerweis.visibility = "visible";
}
function Verstecken() {
  document.BildAnzeige.visibility = "hidden";
  document.VersteckenVerweis.visibility = "hidden";
  document.ZeigenVerweis.visibility = "visible";
}
//-->
</script>
<style type="text/css">
<!--
#ZeigenVerweis {
position:absolute; left:100; top:140; width:400; height:30; visibility:visible; }
#VersteckenVerweis {
position:absolute; left:100; top:140; width:400; height:30; visibility:hidden; }
#BildAnzeige {
position:absolute; left:100; top:170; width:400; height:230; visibility:hidden; }
-->
</style>
</head><body>
<div id="ZeigenVerweis">
<a href="javascript:Zeigen()"><b>Zeig das Bild an</b></a>
</div>
<div id="VersteckenVerweis">
<a href="javascript:Verstecken()"><b>Versteck das Bild</b></a>
</div>
<div id="BildAnzeige">
```

```

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

Das Beispiel bewirkt, dass beim Anklicken eines Verweises ein Bild angezeigt wird, das zunächst nicht sichtbar ist. Gleichzeitig erscheint ein neuer Verweis, der das Ausblenden des Bildes erlaubt. Auf diese Weise lässt sich das Bild immer wieder ein- und ausblenden.



21.15: Simulierte Layer: Ein Mausklick zeigt das Bild im `div`-Bereich an, der nächste lässt es verschwinden.

In dem Beispiel werden dazu innerhalb des Dateikörpers drei Bereiche mit `<div>...</div>` definiert. Alle `div`-Bereiche erhalten CSS-Eigenschaften zur Positionierung, und zwar im Dateikopf in dem `style`-Bereich. Ohne solche Angaben ist bei Netscape kein dynamischer Zugriff auf das entsprechende Element möglich. Dass die CSS-Angaben zentral definiert werden, hat seine Ursache darin, dass einige Netscape 4-Versionen mit Angaben direkt im einleitenden `<div>`-Tag Probleme haben.

Die beiden ersten `<div>`-Bereiche im Beispiel haben die gleichen Angaben zur Positionierung. Sie unterscheiden sich lediglich durch die Wertzuweisung bei der Angabe `visibility`. Dadurch wird bewirkt, dass beide Bereiche an der gleichen Stelle angezeigt werden, der eine Bereich zunächst jedoch ausgeblendet bleibt, während der andere angezeigt wird. Der dritte `<div>`-Bereich ist der Behälter für das anzuzeigende Bild. Er ist zunächst auch nicht sichtbar.

Die beiden ersten Bereiche enthalten jeweils einen Verweis. Beim Anklicken des Verweises wird jeweils eine JavaScript-Funktion aufgerufen, die im Kopf der Datei in einem JavaScript-Bereich definiert ist. Der Bereich, der zunächst angezeigt wird, enthält einen Verweis, bei dessen Anklicken die Funktion `Zeigen()` aufgerufen wird.

Die Funktion `Zeigen()` tut nichts anderes, als die Anzeigeverhältnisse unter den drei `<div>`-Bereichen zu ändern. Der erste Bereich, über dessen Verweis die Funktion aufgerufen wurde, wird mit `document.ZeigenVerweis.visibility="hidden"` auf unsichtbar gesetzt. Der zweite Bereich, der mit dem anderen Verweis, wird dagegen auf sichtbar gesetzt. Auch der Bereich mit dem Bild wird sichtbar. Nun werden der zweite Verweis (»Versteck das Bild«) und das Bild selbst angezeigt. Beim Klicken auf den Verweis wird die andere JavaScript-Funktion `Verstecken()` aufgerufen. Diese Funktion arbeitet so wie die Funktion `Zeigen()`, nur umgekehrt.

Der Zugriff auf die `<div>`-Bereiche wird dadurch möglich, dass bei der Definition der `<div>`-Bereiche mit `id=` ein Name vergeben wird. Über `document.BereichName` ist dann der Zugriff auf Eigenschaften wie `visibility`, `top`, `left`, `width`, `height` oder `clip` möglich. So lassen sich praktisch alle Aufgaben lösen, ohne das `<layer>`-Tag einzusetzen.

**Beachten Sie:** Ein `layer`-Element lässt sich nur simulieren, wenn der `div`-Bereich absolut positioniert ist. Sie können, wie im Beispiel, statt der `visibility`-Eigenschaften `hide` und `show` die Eigenschaften `hidden` und `visible` verwenden, da Netscape 4 diese Angaben ebenfalls korrekt interpretiert.

### 21.4.8 Das Event-Objekt

Das `event`-Objekt von JavaScript erlaubt es, auf Anwendereignisse zu reagieren. Es stellt eine Erweiterung der klassischen Event-Handler dar. Das folgende Beispiel demonstriert die Möglichkeiten des `event`-Objekts. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
layer { font-size:10pt; font-family:Arial }
layer.Kopf { color:#FFFFFF; padding:1mm; cursor:hand; }
layer.Text { color:#000000; padding-top:30px; padding-left:1mm; padding-right:1mm }
-->
</style>
</head><body>

<layer id="Gruen" class="Text" bgcolor="#99FF99" top="170" left="10" width="200"
height="180">
<layer id="Gruenkopf" class="Kopf" bgcolor="#000000" top="0" left="0" width="200"
height="25">
Titelleiste des grünen Layers
</layer>
Klicken Sie mal mit der Maus in die Titelleiste dieses Layers,
halten Sie die Maustaste gedrückt und ziehen Sie dann die Maus an eine andere
Stelle.
</layer>
<layer id="Gelb" class="Text" bgcolor="#FFFF66" top="10" left="310" width="300"
height="120">
<layer id="Gelbkopf" class="Kopf" bgcolor="#000000" top="0" left="0" width="300"
height="25">
```

```

Titelleiste des gelben Layers
</layer>
Klicken Sie mal mit der Maus in die Titelleiste dieses Layers,
halten Sie die Maustaste gedrückt und ziehen Sie dann die Maus an eine andere
Stelle.
</layer>

<script type="text/javascript">
<!--
Gruen = document.layers['Gruen'];
Gruentitel = document.layers['Gruen'].document.layers['Gruenkopf'].document;

Gelb = document.layers['Gelb'];
Gelbtitel = document.layers['Gelb'].document.layers['Gelbkopf'].document;

Gruentitel.captureEvents(Event.MOUSEUP|Event.MOUSEDOWN|Event.MOUSEDRAG);
Gruentitel.onmousedown=StartGruen;
Gruentitel.onmouseup=EndGruen;
document.layers['Gruen'].document.layers['Gruenkopf'].onload=GruenReset;

Gelbtitel.captureEvents(Event.MOUSEUP|Event.MOUSEDOWN|Event.MOUSEDRAG);
Gelbtitel.onmousedown=StartGelb;
Gelbtitel.onmouseup=EndGelb;
document.layers['Gelb'].document.layers['Gelbkopf'].onload=GelbReset;

function GruenReset()
{ Gruentitel.captureEvents(Event.MOUSEUP|Event.MOUSEDOWN|Event.MOUSEDRAG); }
function GelbReset()
{ Gelbtitel.captureEvents(Event.MOUSEUP|Event.MOUSEDOWN|Event.MOUSEDRAG); }

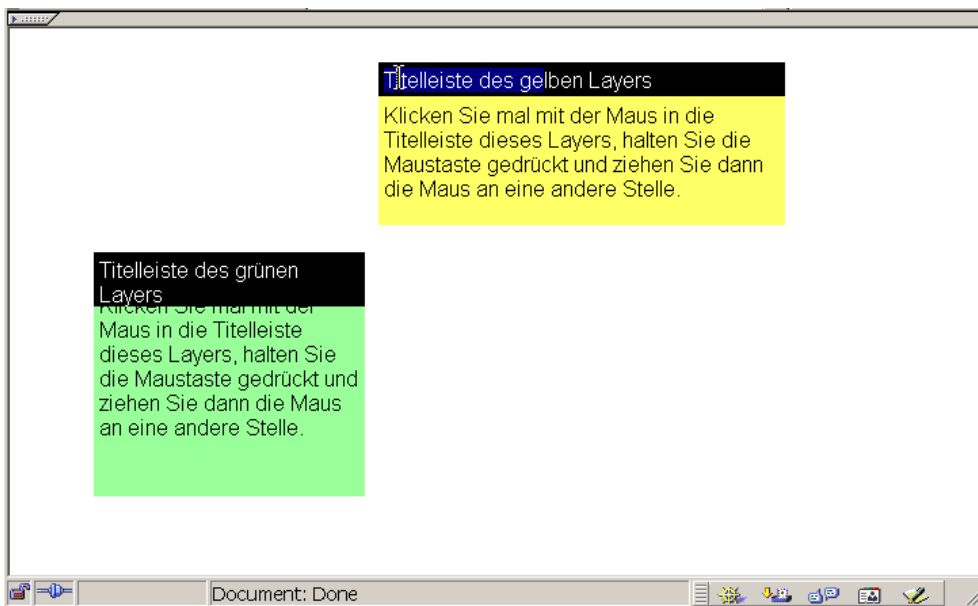
function StartGruen()
{ Gruentitel.captureEvents(Event.MOUSEMOVE); Gruentitel.onmousemove=ZiehGruen; }
function StartGelb()
{ Gelbtitel.captureEvents(Event.MOUSEMOVE); Gelbtitel.onmousemove=ZiehGelb; }

function ZiehGruen(Ereignis)
{ Gruen.left = Ereignis.pageX; Gruen.top = Ereignis.pageY; }
function ZiehGelb(Ereignis)
{ Gelb.left = Ereignis.pageX; Gelb.top = Ereignis.pageY; }

function EndGruen()
{ Gruentitel.onmousemove=0; Gruentitel.releaseEvents(Event.MOUSEMOVE); }
function EndGelb()
{ Gelbtitel.onmousemove=0; Gelbtitel.releaseEvents(Event.MOUSEMOVE); }
//-->
</script>
</body></html>

```

Im Beispiel werden insgesamt vier Layer definiert, davon zwei »äußere« (mit `id="Gruen"` bzw. `id="Gelb"`) und zu jedem jeweils ein »innerer« (mit `id="Gruenkopf"` bzw. `id="Gelbkopf"`). Die inneren Layer sind so gestaltet, dass sie wie Titelleisten zu den jeweils äußeren Layern wirken, daher auch die `id`-Namen.



21.16: Mit gedrückter Maustaste verschieben Sie die Layer an eine neue Position.

Wie Sie erkennen können, steht nirgendwo bei der Layer-Definition oder innerhalb eines der Layer ein JavaScript-Aufruf. Dennoch sind die beiden Titelleisten »sensitiv«: Wenn der Anwender mit der Maus darauf klickt, die Maustaste gedrückt hält und die Maus bewegt, bewegt sich der gesamte zugehörige »äußere« Layer mit. Der Grund dafür ist das Konzept des `event`-Objekts. Es benötigt keine Event-Handler innerhalb von HTML, um JavaScript-Aktionen anzustoßen.

Der JavaScript-Bereich des obigen Beispiels enthält dafür einige »ungewohnte« Dinge. Er wird unterhalb der Layer definiert, weil gleich bei seinem Einlesen Code ausgeführt wird, der sich auf die definierten Layer bezieht. Die Layer müssen zu diesem Zeitpunkt bereits eingelesen sein, sonst gibt es eine Fehlermeldung.

Die ersten vier Zeilen des JavaScript-Codes im Beispiel speichern einfach nur ziemlich lange Objekt-Eigenschaft-Ketten in Variablen, da diese im weiteren Verlauf häufig notiert werden müssen. Danach folgen jeweils vier Anweisungen, und zwar je für den grünen Layer und für den gelben. Die Methode `captureEvents()` bewirkt, dass Anwenderaktionen überwacht und bei Eintritt gemeldet werden. Die Methode `captureEvents()` erwartet als Parameter einen Wert. Notieren Sie dazu nach dem Prinzip `Event.WELCHER` die Events, die überwacht werden sollen. Die gewünschten Events sind durch bitweises Oder (einfacher Vertikalstrich) zu trennen. Welche Events es neben denen im obigen Beispiel gibt, erfahren Sie in der JavaScript Objekt-Referenz beim Objekt `event` (Kapitel 20.18).

Beim Eintritt von überwachten Ereignissen werden Funktionen aufgerufen. Beachten Sie dabei die etwas merkwürdige Syntax: Mit einer Anweisung wie `Gruentitel.onmousedown=StartGruen` wird die selbst geschriebene Funktion `StartGruen()` aufgerufen, die weiter unten im Script definiert ist. Beim Aufruf dürfen Sie in diesem Fall jedoch keine Klammern hinter dem Funktionsnamen notieren!

Verfolgen Sie nun mal im Beispiel den Fall, dass der Anwender auf die Titelleiste des grünen Layers klickt. Dann wird das Ereignis `MOUSEDOWN` für `Gruentitel` gemeldet. Für diesen Fall (`onmousedown`) wird die Funktion `StartGruen` aufgerufen. Innerhalb dieser Funktion wird wiederum überwacht, ob der Anwender die Maus bei gedrückter Maustaste bewegt. Wenn ja, wird dort nach dem gleichen Schema die Funktion `ZiehGruen` aufgerufen. Obwohl die Funktion keinen Parameter übergeben bekommt, erwartet sie einen – auch das gegen alle sonstige Regeln. Der erwartete Parameter ist aber kein »Wert« wie sonst üblich, sondern einfach ein Name für den aktuellen Event. Im Beispiel wurde der Name `Ereignis` gewählt. Unter dem vergebenen Namen lassen sich Eigenschaften des `event`-Objekts ansprechen. Da innerhalb der Funktion `ZiehGruen` davon Gebrauch gemacht wird, ist ein solcher Name erforderlich. Aktiv ist die Funktion `ZiehGruen` so lange, wie der Anwender die Maustaste bewegt und nicht loslässt. Während dieser Zeit werden »so oft wie möglich« die Eigenschaften `PageX` (aktuelle Mausposition von links) und `PageY` (aktuelle Mausposition von oben) von `Ereignis` abgefragt. Diese Werte werden dem betroffenen Layer dynamisch als neue Werte für die linke obere Ecke zugewiesen. So entsteht der Bewegungseffekt.

Wenn der Anwender die Maustaste loslässt, wird die Funktion `ZiehGruen` sofort verlassen. Da die Event-Überwachung ja noch aktiv ist, wird jetzt das Ereignis `MOUSEUP` gemeldet. Für diesen Fall wird die Funktion `EndGruen` aufgerufen. Diese Funktion enthält zwei Befehle zum »Aufräumen«.

Wenn Sie sonst nichts weiter unternehmen würden, wäre der Layer nach dem ersten Bewegen nicht mehr bewegungsfähig. Damit er immer wieder bewegbar ist, tritt automatisch ein `Load`-Ereignis ein. Dieses wird mit `onload` abgefangen. Dabei wird die Funktion `GruenReset` aufgerufen. Diese Funktion startet die erneute Überwachung der Ereignisse. Analog zu dem Handling für den grünen Layer wird mit dem Handling für den gelben Layer verfahren.

### 21.4.9 Inline-Scripts bei Layern

Netscape bietet eine vereinfachte JavaScript-Syntax an, um Layer-Eigenschaften innerhalb eines Layers dynamisch zu ändern. Ein Beispiel:

```
<html><head><title>Test</title>
<style type="text/css">
<!--
layer { padding:1mm; }
p.Verweis { font-size:10pt; font-weight:bold; font-family:Arial; color:yellow; }
p.Text { font-size:10pt; font-family:Arial; color:white; }
-->
</style>
</head><body>
<layer class="Kopf" bgcolor=#000000 top="175" left="50" width="500" height="200"
clip="0,0,500,25" onMouseover="Zeigen()" onMouseout="Verstecken()">
```

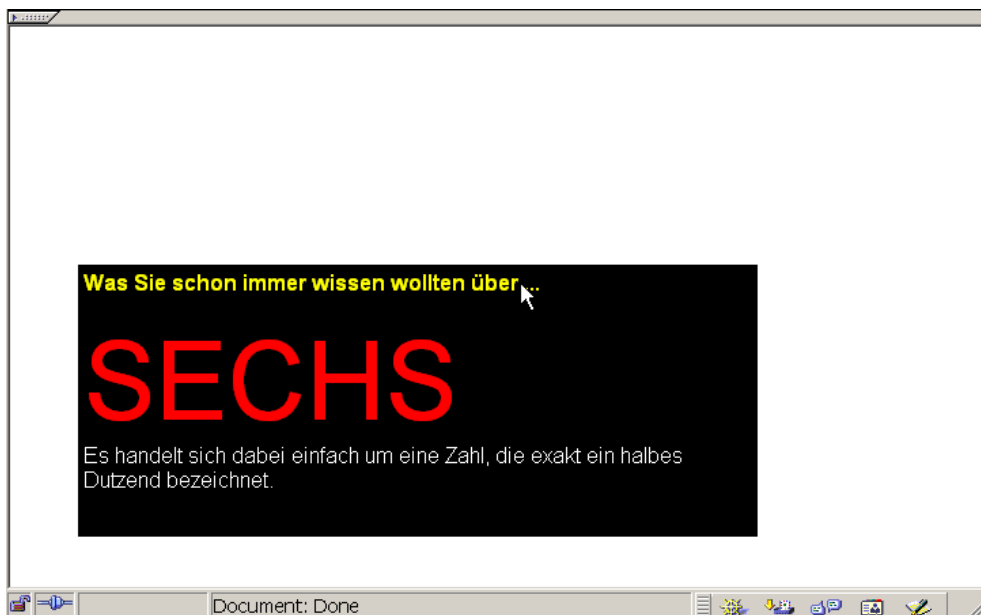


```

<p class="Verweis">Was Sie schon immer wissen wollten über ...</p>
<p class="Text"><span style="color:red; font-size:48pt">SECHS</span><br>
Es handelt sich dabei einfach um eine Zahl, die exakt ein halbes Dutzend
bezeichnet.</p>
<script type="text/javascript">
<!--
function Zeigen() { clip.bottom = 200; }
function Verstecken() { clip.bottom = 25; }
//-->
</script>
</layer>
</body></html>

```

Im Beispiel wird ein Layer definiert. Mit einer Angabe zu `clip=` wird der Anzeigebereich des Layers jedoch zunächst beschnitten, und zwar so, dass gerade der erste Textabsatz sichtbar ist. Wenn der Anwender mit der Maus über den sichtbaren Layer-Bereich fährt, wird die JavaScript-Funktion `Zeigen()` aufgerufen. Diese Funktion verändert den Clip-Bereich des Layers so, dass er in ganzer Größe angezeigt wird. Wenn die Maus den Layer-Bereich wieder verlässt, wird die Funktion `Verstecken()` aufgerufen. Diese Funktion ändert den Clip-Bereich wieder auf den Anfangswert.



21.17: Beim Überfahren mit der Maus klappt das Layer auf.

Wichtig ist bei dieser Form, dass der JavaScript-Bereich innerhalb des Layers definiert wird. Dann brauchen Sie sich nicht um die korrekte Adressierung des Layers mit Indexnummern oder Namen zu kümmern. Sie können einfach Anweisungen wie `clip.bottom=[Wert]` (wie im Beispiel) oder Anweisungen wie `top=`, `left=` oder `bgColor=` notieren, um Wirkungsweise und

Aussehen des Layers zu verändern. Welche Eigenschaften Sie ändern können, entnehmen Sie der Referenz zum Objekt `layers` (Kapitel 20.16).

## 21.5 Datenanbindung (nur Microsoft)

### 21.5.1 Allgemeines zur Datenanbindung

Der MS Internet Explorer ab Version 4.x erlaubt es, in kontrollierter Form Daten aus einer externen Datenquelle zu holen und in HTML-Elementen dynamisch anzuzeigen. So ist beispielsweise eine Tabelle denkbar, in der sich der Anwender bei jedem Klick auf einen entsprechenden Button die Bundesliga-Endtabellen der Spielsaisons von 1960 bis heute anzeigen lassen kann. Bei jedem Klick werden aus einer externen Datei eine Reihe von Daten ausgelesen und dynamisch in die Anzeigtabelle übernommen.

Microsoft hat das Konzept der Datenanbindung sehr flexibel ausgelegt. Es gibt diverse Möglichkeiten, auf externe Datenquellen zuzugreifen. Der Zugriff auf eine einfache Textdatei ist nur eine davon. Es besteht ebenso die Möglichkeit, etwa über die Datenbankabfragesprache SQL auf eine Datenbank zuzugreifen, die SQL-Befehle verarbeiten kann. Für die verschiedenen Möglichkeiten der Datenanbindung gibt es verschiedene **Datenmodelle**. Für jedes der Datenmodelle gibt es ein bestimmtes ActiveX-Control oder ein Java-Applet, das Sie in Ihrer HTML-Datei ansprechen müssen, um die Datenanbindung damit zu realisieren. Die Datenmodelle und deren Einbindung in HTML werden auf dieser Seite vorgestellt.

Die Vorteile der Datenanbindung liegen auf der Hand: Daten können mit einer dafür geeigneten Anwendung gepflegt und gespeichert werden. Eine HTML-Datei kann auf immer aktuelle Daten zugreifen, ohne selbst geändert werden zu müssen. Für das »Holen« der Daten gibt es kein vorgeschriebenes Protokoll. Die Datenanbindung funktioniert also auf allen Umgebungen, ob auf einem WWW-Server, lokal auf einer PC-Festplatte oder auf einer CD-ROM (zumindest auf das einfachste Datenmodell trifft dies zu). Es ist kein zusätzliches CGI-Script erforderlich, das als Vermittler zwischen HTML-Datei und Datenbestand für jede Interaktion zwischen Anwender und Datenbank immer wieder aufgerufen werden muss. Das Know-how hat der Browser, und die Steuerung der Datenanbindung wird direkt in der HTML-Datei notiert. Durch dynamisches Austauschen der Inhalte von HTML-Elementen, in denen die Daten angezeigt werden, ist kein neuer Aufbau der WWW-Seite erforderlich.

Aus heutiger Sicht muss das Modell der clientseitigen Datenanbindung, das Microsoft mit dem Internet Explorer 4.0 eingeführt hat, jedoch kritisch betrachtet werden. Der Grund ist einerseits, dass es auf extrem proprietären Konzepten beruht, vor allem was die Einbindung der allgemein unbeliebten ActiveX-Controls betrifft. Und andererseits haben sich Web-Server mittlerweile so stark durchgesetzt, dass es in den meisten Fällen – auch lokal – sinnvoller ist, serverseitige CGI-Scripts zum Handling der Datenverarbeitung einzusetzen.

**Hinweis:** Die Erläuterungen zu den Datenmodellen weiter hinten in diesem Kapitel sind zum Teil sehr technisch. Das ist leider nicht zu ändern, da die Materie ziemlich komplex ist. Die Erläuterungen zu den Datenmodellen richten sich auch nur an Spezialisten, die wissen, was mit den entsprechenden Begriffen anzufangen ist.

### 21.5.2 Beispiel: Datei dynamisch in Tabelle einbinden

Die hier beschriebene Vorgehensweise erläutert, wie Sie in einer Tabelle einzelne Datensätze aus einer externen Textdatei anzeigen können. Dabei wird das Datenmodell Tabular Data Control (TDC) verwendet. Mit Hilfe von Buttons kann der Anwender den nächsten oder vorhergehenden Datensatz in die Tabelle laden.



| Adressen      |                  |
|---------------|------------------|
| Name:         | Rauhbein         |
| Vorname:      | Eberhard         |
| Straße:       | Traumstr. 13     |
| Postleitzahl: | 56097            |
| Wohnort:      | Höllenbachhausen |
| Telefon:      | 068347-582374    |
| E-Mail:       | -                |

21.18: Die Web-Seite im Beispiel greift auf eine externe Adressdatei im Textformat zu.

Beispiel Teil 1 – externe Datei *adressen.txt* mit Daten:

```
Name|Vorname|Strasse|PLZ|Ort|Tel|EMail
Berger|Hans|Pfisterstr.11|38487|Grubenhausen|04987-4445631|berger@kabumm.de
Heidenfroh|Martina|Mittelweg 123|43985|Hatzenau|04583-2398474|mh@heidenfroh.com
Rauhbein|Eberhard|Traumstr.13|56097|Höllenbachhausen|068347-582374|-
Truthahn|Gerlinde|Aufplatz 1|93722|Sündenstadt|032987-385750|gerlinde@metawelt.net
```

Die externe Datei ist in diesem Beispiel eine einfache Textdatei, z.B. eine Datei mit dem Namen *adressen.txt*. Die Datei ist so aufgebaut, dass jede Zeile einen Datensatz darstellt. Die einzelnen Felder eines Datensatzes werden in diesem Beispiel durch das Zeichen »|« getrennt. Es könnte auch ein anderes Zeichen sein. Das Zeichen »|« wurde hier gewählt, weil es sehr selten ist und in den Daten selbst nicht vorkommt. In der ersten Zeile der Datei stehen keine Daten, sondern Feldnamen. Der dritte Feldname lautet beispielsweise *Strasse*. In den eigentlichen Datenzeilen darunter stehen dann in entsprechender Reihenfolge die Daten. So hat das dritte Feld der ersten Datenzeile den Wert *Pfisterstr.11*. Dies ist also ein Wert für den in der ersten Zeile an dritter Stelle vergebenen Feldnamen *Strasse*.

Textdateien dieser Art sind nicht besonders komfortabel, wenn man sie selbst in einem Texteditor bearbeiten will. Doch solche Textdateien lassen sich mit fast allen Tabellenkalkulations- und Datenbankprogrammen erzeugen. Meist ist dafür der Datentyp »kommabegrenzt« (»comma delimited«) mit der Standard-Dateiendung *.csv* vorgesehen. Bessere Programme erlauben es dabei auch, ein selbst definierbares Feldbegrenzerzeichen zu vergeben.

## Beispiel Teil 2 – HTML-Datei mit Datenzugriff:

```

<html><head><title>Test</title></head>
<body>
<object id="Adressen" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name="DataURL" value="adressen.txt">
  <param name="UseHeader" value="true">
  <param name="FieldDelim" value="|">
</object>
<form action="">
<table id="Anzeigetabelle" datasrc="#Adressen" datapagesize="1" width="100%"
border="1">
<colgroup><col width="200"></colgroup>
<tr>
<td bgcolor="#EEEEEE"><b>Adressen</b></td>
<td bgcolor="#EEEEEE" align="right">
<input type="button" value=" < "
onclick="document.all.Anzeigetabelle.previousPage();">
<input type="button" value=" > "
onclick="document.all.Anzeigetabelle.nextPage();">
</td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>Name:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="Name"></span></td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>Vorname:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="Vorname"></span></td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>Straße:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="Strasse"></span></td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>Postleitzahl:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="PLZ"></span></td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>Wohnort:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="Ort"></span></td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>Telefon:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="Tel"></span></td>
</tr><tr>
<td bgcolor="#EEEEEE" valign="top"><b>E-Mail:</b></td>
<td bgcolor="#FFFFFF0" valign="top"><span datafld="EMail"></span></td>
</tr></table></form>
</body>
</html>

```

Um eine externe Datenquelle in einer HTML-Datei nutzen zu können, müssen Sie zunächst die Quelle und diverse Angaben zu deren Aufbau notieren. Dazu brauchen Sie das `<object>`-Tag.

In dem einleitenden `<object>`-Tag sind zwei Attribute erforderlich: Mit der Angabe `id=` vergeben Sie einen Namen, unter dem Sie später auf die Daten zugreifen können. Der Name sollte passend sein, im obigen Beispiel etwa "Adressen". Als zweiten Parameter müssen Sie das ActiveX-Control referenzieren, das den gesamten Zugriff auf die externe Datei abwickelt. Bei einfachen Textdateien mit zeilenweisen Datensätzen müssen Sie dabei immer die Angabe `classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"` notieren. Das ActiveX-Control mit dieser Identifikationsnummer ist im Lieferumfang des MS Internet Explorers ab Version 4.x enthalten – Sie brauchen sich um nichts weiter zu kümmern. Genau dabei liegt allerdings der Knackpunkt des gesamten Konzepts: Ohne solche ActiveX-Controls läuft nichts.

Innerhalb des `<object>`-Tags notieren Sie Angaben zu Ort und Beschaffenheit der externen Datei.

- Mit `<param name="DataURL" value=` können Sie die gewünschte Datei ansprechen. Hinter `value=` notieren Sie den Dateinamen. Wenn sich die Datei im gleichen Verzeichnis befindet, genügt der einfache Dateiname. Ansonsten gelten die gleichen Regeln wie bei Verweisen in HTML.
- Mit `<param name="UseHeader" value="true">` weisen Sie den Browser an, die erste Zeile der externen Datei nicht als Datenzeile zu interpretieren, sondern als Zeile für die Definition der Feldnamen.
- Mit `<param name="FieldDelim" value="|">` geben Sie an, welches das Feldtrennzeichen sein soll. Im Beispiel ist dies das Zeichen "|".

Damit sind alle Voraussetzungen geschaffen, um die Daten aus der externen Datei dynamisch in HTML anzuzeigen. Die Anzeige der Daten erfolgt in einer Tabelle. Das einleitende `<table>`-Tag schafft die nötigen Voraussetzungen, um die Daten in kontrollierter Form anzuzeigen. Zunächst wird in der Tabelle mit `id=` ein Name vergeben (im Beispiel: `Anzeigetabelle`). Dieser Name wird bei den Buttons benötigt, die innerhalb der Tabelle das Blättern innerhalb der Datensätze ermöglichen.

Die nächste Angabe im `<table>`-Tag ist `datasrc=` (*data source, Datenquelle*). Damit nehmen Sie Bezug auf die externe Datendatei. An dieser Stelle dürfen Sie jedoch nicht die Datendatei selbst angeben, sondern Sie müssen denjenigen Namen angeben, den Sie zuvor im `<object>`-Tag bei `id=` vergeben haben. Und zwar mit einem vorangestellten Gatterzeichen `#`. Dadurch stellen Sie den Bezug zwischen Tabelle und Datenquelle her. Es werden dann alle Parameter berücksichtigt, die Sie innerhalb des `<object>`-Tags angegeben haben.

Ebenfalls wichtig ist die Angabe `datapagesize=` (*Datenseitengröße*) im `<table>`-Tag. Damit geben Sie an, wie viele Datensätze aus der externen Datei gleichzeitig angezeigt werden sollen. Wenn Sie diese Angabe weglassen, macht der MS Internet Explorer die Tabelle automatisch so lang, dass alle Datensätze auf einmal darin dargestellt werden. Im obigen Beispiel wird `datapagesize="1"` angegeben. Dadurch wird immer genau ein Datensatz angezeigt.

Innerhalb der Tabelle können Sie erkennen, wie ein Datensatz aus der externen Datenquelle darin angezeigt wird. Die Tabelle selbst können Sie nach Ihren Wünschen frei gestalten. Wichtig ist nur zu wissen, wie der Bezug zu einem Datensatz der externen Datei hergestellt wird. Im obigen Beispiel geschieht dies innerhalb entsprechender Tabellenzellen mit Hilfe des `<span>`-Tags. Mit einem Befehl wie `<span datafld="PLZ"></span>` weisen Sie den Browser an, in der betreffenden Tabellenzelle den aktuellen Inhalt des Feldes "PLZ" aus der externen Datenquelle anzuzeigen (*datafld = data field = Datenfeld*). Dabei muss der Wert, den Sie hinter der Angabe

`datafld=` notieren, mit dem Feldnamen übereinstimmen, den Sie in der ersten Zeile der externen Datendatei vergeben haben. Beachten Sie dabei auch die exakte Schreibweise.

Neben der Angabe `datafld=` können Sie in so einem Datenanzeigebereich auch noch die Angabe `dataformatas=` (*formatiere Daten als*) notieren. Im obigen Beispiel wird davon kein Gebrauch gemacht. Benutzen Sie `dataformatas=`, wenn die Daten in der externen Datei selbst HTML-Tags enthalten. Mit `dataformatas="html"` erzwingen Sie dann, dass die Tags aus der externen Datei als HTML-Tags interpretiert werden.

Da im obigen Beispiel mit `datapagesize="1"` festgelegt wurde, dass immer nur ein Datensatz angezeigt wird, erscheint in der Tabelle zunächst der Inhalt der ersten Datenzeile der externen Datei. Natürlich soll der Anwender nun in den Datensätzen blättern können. Dazu werden Buttons zum Vor- und Zurückblättern definiert. Im Beispiel befinden sich diese Buttons in der ersten Tabellenzeile. Sie könnten aber auch an einer anderen Stelle stehen.

Mit `<input type=button value=" < " onclick="document.all.Anzeigetabelle.previousPage();">` wird ein Button definiert, bei dessen Anklicken der vorhergehende Datensatz aufgerufen wird. Mit `document.all.Anzeigetabelle` greifen Sie dabei auf die Tabelle zu, in der die Daten angezeigt werden (`Anzeigetabelle` ist dabei der Name, der in der Tabelle mit `id=` vergeben wurde). Mit `<input type=button value=" > " onclick="document.all.Anzeigetabelle.nextPage();">` wird der Button definiert, der zum nächsten Datensatz springt. Beachten Sie, dass Sie nicht kontrollieren müssen, ob der Anwender gerade den ersten oder letzten Datensatz anzeigt. Es gibt keine Fehlermeldungen, wenn der Anwender versucht, einen nicht vorhandenen Datensatz aufzurufen. Der Button hat dann einfach keine Wirkung.

### 21.5.3 Beispiel: Daten durchsuchbar machen

In diesem Beispiel wird gezeigt, wie Sie dem Anwender ermöglichen können, mit Hilfe eines Suchbegriffes nur diejenigen Datensätze anzuzeigen, in denen der gesuchte Begriff vorkommt. Das Beispiel zeigt auch, wie Sie mit Hilfe von Scripts auf die HTML-Elemente zur Datenanbindung zugreifen können.

Beispiel Teil 1 – externe Datei *woerterbuch.txt* mit Daten:

```
Deutsch;Englisch;Franzoesisch;Italienisch;Spanisch
Rechner;computer;ordinateur;computer, calcolatore;ordenador
Maschine;machine (s);machine;macchina;máquina
Personalcomputer;personal computer;ordinateur personnel (m);computer
personale;ordenador personal
arbeitender Rechner;active computer;ordinateur active;computer attivo;ordenador
activado
... usw. ...
```

Die Datendatei dieses Beispiels zeigt ein mehrsprachiges Fachwörterbuch. In der ersten Zeile werden die Sprachen definiert. Unterhalb davon stehen die Wörter in den jeweiligen Sprachen. Die Einträge zu den Sprachen sind durch Semikola (;) getrennt. Wenn in einer Sprache mehrere Wörter in Frage kommen, werden diese durch Kommata getrennt, so wie in der zweiten Zeile beim italienischen Wort für Rechner: `computer, calcolatore`.

Die durchsuchbare Datenbank enthält 1000 Einträge. Suchen Sie nach Begriffen in einer oder mehreren Sprachen. Bei der Suche in mehreren Sprachen werden die Eingaben logisch durch UND verknüpft. Sie dürfen auch das Wildcardzeichen \* für Teilausdrücke verwenden. Suchen Sie z.B. mal im Feld (Deutsch) nach **Arbeits\*** oder im Feld (Italienisch) nach **\*ore** oder im Feld (Englisch) nach **\*computer**.

Suche nach:

|                                         |                |
|-----------------------------------------|----------------|
| <input type="text" value="Affengriff"/> | (Deutsch)      |
| <input type="text"/>                    | (Englisch)     |
| <input type="text"/>                    | (Franzoesisch) |
| <input type="text"/>                    | (Italienisch)  |
| <input type="text"/>                    | (Spanisch)     |

**0 Suchtreffer!**

**Deutsch: Englisch: Französisch: Italienisch: Spanisch:**

Fertig Arbeitsplatz

21.19: Der Internet Explorer sucht in der Textdatenbank.

## Beispiel Teil 2 – HTML-Datei mit Datenzugriff:

```

<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
function Suchen() {
    var Datensaeetze = 1000;
    document.all.Anzeigetabelle.style.display = "none";
    var Suchausdruck = "";
    var Suchbegriffe = 0;
    if(document.Eingabe.SuchDeu.value != "") {
        Suchausdruck = Suchausdruck + "Deutsch = " + document.Eingabe.SuchDeu.value;
        Suchbegriffe++;
    }
    if(document.Eingabe.SuchEng.value != "") {
        if(Suchbegriffe > 0)
            Suchausdruck = Suchausdruck + " & Englisch = " +
document.Eingabe.SuchEng.value;
        else
            Suchausdruck = Suchausdruck + "Englisch = " + document.Eingabe.SuchEng.value;
        Suchbegriffe++;
    }
    if(document.Eingabe.SuchFra.value != "") {
        if(Suchbegriffe > 0)
            Suchausdruck = Suchausdruck + " & Franzoesisch = " +
document.Eingabe.SuchFra.value;

```

```

    else
        Suchausdruck = Suchausdruck + "Franzoesisch = " +
document.Eingabe.SuchFra.value;
        Suchbegriffe++;
    }
    if(document.Eingabe.SuchIta.value != "") {
        if(Suchbegriffe > 0)
            Suchausdruck = Suchausdruck + " & Italienisch = " +
document.Eingabe.SuchIta.value;
        else
            Suchausdruck = Suchausdruck + "Italienisch = " +
document.Eingabe.SuchIta.value;
        Suchbegriffe++;
    }
    if(document.Eingabe.SuchSpa.value != "") {
        if(Suchbegriffe > 0)
            Suchausdruck = Suchausdruck + " & Spanisch = " +
document.Eingabe.SuchSpa.value;
        else
            Suchausdruck = Suchausdruck + "Spanisch = " + document.Eingabe.SuchSpa.value;
        Suchbegriffe++;
    }
    document.all.Begriffe.object.Filter = Suchausdruck;
    document.all.Begriffe.Reset();
    document.all.Anzeigetabelle.dataSrc = "#Begriffe";
    if(document.all.Begriffe.recordset.recordCount < Datensaeetze) {
        document.all.Anzeigetabelle.dataPageSize = Begriffe.recordset.recordCount;
        document.all.Treffer.innerText = Begriffe.recordset.recordCount + "
Suchtreffer!";
        document.all.Anzeigetabelle.style.display = "block";
    }
    else
        document.all.Treffer.innerText = "0 Suchtreffer!";
}
//-->
</script>
</head>
<body>
<object id="Begriffe" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name="DataURL" value="woerterbuch.txt">
  <param name="UseHeader" value="true">
  <param name="Filter" value="">
  <param name="FieldDelim" value=";">
</object>

<p>Die durchsuchbare Datenbank enthält 1000 Einträge. Suchen Sie nach
Begriffen in einer oder mehreren Sprachen. Bei der Suche in mehreren Sprachen
werden die Eingaben logisch durch UND verknüpft. Sie dürfen auch das
Wildcardzeichen <b>*</b> für Teilausdrücke verwenden. Suchen Sie z.B. mal im Feld

```



```

(Deutsch) nach <b>Arbeits*</b> oder im Feld (Italienisch) nach
<b>*ore</b> oder im Feld (Englisch) nach <b>*computer</b>.</p>

<form name="Eingabe" action="">
Suche nach:<br>
<input name="SuchDeu" type="text" width="40"> (Deutsch)<br>
<input name="SuchEng" type="text" width="40"> (Englisch)<br>
<input name="SuchFra" type="text" width="40"> (Franzoesisch)<br>
<input name="SuchIta" type="text" width="40"> (Italienisch)<br>
<input name="SuchSpa" type="text" width="40"> (Spanisch)<br>
<input type="button" style="width:150px; height:25px;" value="Suche starten"
onClick="Suchen()">
</form>

<table border="1" cellpadding="1" cellspacing="0"><tr><td>
<table width="100%">
<tr>
<td valign="top" colspan="5">
<span id="Treffer" style="color:#0000CC; font-weight:bold;"> </span>
</td>
</tr>
<tr>
<td bgcolor="#EEEEEE" valign="top" width="20%"><b>Deutsch:</b></td>
<td bgcolor="#EEEEEE" valign="top" width="20%"><b>Englisch:</b></td>
<td bgcolor="#EEEEEE" valign="top" width="20%"><b>Französisch:</b></td>
<td bgcolor="#EEEEEE" valign="top" width="20%"><b>Italienisch:</b></td>
<td bgcolor="#EEEEEE" valign="top" width="20%"><b>Spanisch:</b></td>
</tr>
</table>
<table id="Anzeigetabelle" style="display:none" width="100%">
<tr>
<td bgcolor="#FFFFE0" valign="top" width="20%"><span datafld="Deutsch"></span></td>
<td bgcolor="#FFFFE0" valign="top" width="20%"><span
datafld="Englisch"></span></td>
<td bgcolor="#FFFFE0" valign="top" width="20%"><span
datafld="Franzoesisch"></span></td>
<td bgcolor="#FFFFE0" valign="top" width="20%"><span
datafld="Italienisch"></span></td>
<td bgcolor="#FFFFE0" valign="top" width="20%"><span
datafld="Spanisch"></span></td>
</tr>
</table>
</td></tr></table>
</body>
</html>

```

Das Beispiel zeigt die vollständige HTML-Datei zur Realisierung einer durchsuchbaren Datenbank. Innerhalb der HTML-Datei wird wie im Beispiel *Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2) wieder das `<object>`-Tag notiert, um die Verbindung zu der externen Daten-datei herzustellen. Es wird jedoch ein neuer Parameter angegeben: `<param name="Filter"`

`value=""`> Mit dem Parameter `Filter` können Sie die Datensätze, die aus der Datendatei »geholt« werden, begrenzen. Dazu können Sie bei `value=` Einschränkungskriterien angeben. Bei der Objektdefinition im obigen Beispiel wird `value=""` notiert. Das bedeutet, es gibt keine Einschränkungen. Alle Datensätze werden gefunden.

In dem längeren Script, das im Kopf der HTML-Datei steht, wird der Wert für `Filter` dynamisch verändert. Dadurch wird die Anzeige auf bestimmte Datensätze eingeschränkt, nämlich solche, die den Filterkriterien entsprechen. Die Kriterien für den Filter soll der Anwender durch seine Suchanfrage selbst bestimmen können. Im obigen Beispiel wird dazu unterhalb des `<object>`-Tags ein Formular definiert (`<form name="Eingabe" action="">`). Das Formular hat fünf Eingabefelder, eines für jede der fünf Sprachen des mehrsprachigen Wörterbuchs. Mit einem anklickbaren Button kann der Anwender die Suche abschicken. Beim Anklicken des Buttons wird die Funktion `Suchen()` aufgerufen, die in dem JScript-Bereich im Dateikopf definiert ist (`onClick="Suchen()"`).

Innerhalb der Funktion `Suchen()` wird zunächst eine Variable `Datensaetze` mit einem Wert vorgelegt (1000). Das empfiehlt sich bei größeren Datenbeständen. Denn per Voreinstellung liefert der MS Internet Explorer, wenn er keine Datensätze findet, die dem Suchkriterium entsprechen, alle Datensätze zurück. Angenommen, Sie haben 5000 Datensätze. Dann würde eine Tabelle mit 5000 Zeilen angezeigt. Da dies im Aufbau ziemlich lange dauert, ist eine Begrenzung sinnvoll.

Die Variable `Suchausdruck` soll die Anwenderkriterien zum Durchsuchen des Datenbestandes aufnehmen. Dabei ist es erforderlich, die Angaben aus den Formularfeldern in eine Zeichenkette zu konvertieren, die den Regeln für den Parameter `Filter` entsprechen.

Folgende wichtige Regeln gelten für Werte, die bei `Filter` erlaubt sind (diese Regeln gelten sowohl bei Wertzuweisungen innerhalb des `<object>`-Tags als auch bei dynamischen Wertzuweisungen innerhalb von Scripts):

- Mit `Feldname = Wert` werden nur Datensätze gefunden, bei denen das Feld `Feldname` exakt den Wert `Wert` enthält und nicht mehr und nicht weniger. Bei `Name = Meyer` werden also nur Meyers gefunden, keine Meyerbohms.
- Mit `Feldname <> Wert` werden Datensätze gefunden, bei denen der Wert `Wert` im Feld `Feldname` vorkommt – er kann auch exakt damit übereinstimmen. Bei `Name <> Meyer` werden also Meyers und Meyerbohms gefunden.
- Mit `Feldname > Wert` werden Datensätze gefunden, bei denen der Wert von `Feldname` größer ist als `Wert`. Bei `Name > Y` werden also nur Namen gefunden, die mit Z anfangen.
- Mit `Feldname >= Wert` werden Datensätze gefunden, bei denen der Wert von `Feldname` größer oder gleich ist im Vergleich zu `Wert`. Bei `Name >= Y` werden also nur Namen gefunden, die mit Y oder Z anfangen.
- Mit `Feldname < Wert` werden Datensätze gefunden, bei denen der Wert von `Feldname` kleiner ist als `Wert`. Bei `Name < D` werden also nur Namen gefunden, die mit A,B oder C anfangen.
- Mit `Feldname <= Wert` werden Datensätze gefunden, bei denen der Wert von `Feldname` kleiner oder gleich ist im Vergleich zu `Wert`. Bei `Name <= D` werden also Namen gefunden, die mit A,B,C oder D anfangen.

- An jeder Stelle ist ein Platzhalterzeichen erlaubt. Dies ist ein Stern-Zeichen (\*). Es steht für kein, ein oder beliebig viele Zeichen.
- Sie können auch zwei oder mehrere solcher einfachen Ausdrücke mit den Zeichen & (für UND) oder | (für inklusives ODER) logisch verknüpfen. Wenn Sie mehrere Ausdrücke verknüpfen und dabei sowohl UND also auch ODER verwenden, müssen Sie Klammern verwenden, um eine eindeutige Bewertung des gesamten Kriteriums sicherzustellen. Ein komplexer Ausdruck dieser Art ist etwa `Name > M & PLZ = 8*`.

Im obigen Script wird mit diversen `if`-Abfragen aus den Eingabefeldern des Formulars ein gültiger Zuweisungswert für `Filter` konstruiert. Dabei wird nur die einfachste Bedingung mit dem Gleichheitszeichen eingesetzt. Da der Anwender auch selbst das Stern-Zeichen als Platzhalterzeichen in einem Eingabefeld eingeben kann, genügt diese einfache Suche für viele Fälle. Wenn Sie dem Anwender komplexere Suchabfragen ermöglichen wollen, müssen Sie diesen Teil des Scripts entsprechend aufwändiger gestalten.

Im unteren Teil der Funktion `Suchen()` steht dann der Befehl:

```
document.all.Begriffe.object.Filter = Suchausdruck;
```

Damit wird dem Parameter `Filter` des `<object>`-Tags der Suchausdruck zugewiesen, der zuvor ermittelt und in der Variablen `Suchausdruck` gespeichert wurde. Der nächste Befehl ist ebenfalls von entscheidender Bedeutung:

```
document.all.Begriffe.Reset();
```

Damit veranlassen Sie den Browser, die Anzahl der anzuzeigenden Datensätze neu zu ermitteln. Da mittlerweile ein Filter bestimmt wurde, findet der Browser nun nur noch Datensätze, die den Bedingungen des Filters entsprechen. Auf die übrigen Anweisungen des Scripts und die Elemente in der HTML-Datei wird hier nicht näher eingegangen. Es handelt sich dabei um einfaches dynamisches HTML nach den Regeln des MS Internet Explorers 4.x.

## 21.5.4 HTML-Elemente für die Anzeige von externen Daten

Im Beispiel *Datei dynamisch in Tabelle einbinden* in Kapitel 21.5.2 werden die externen Daten jeweils in einem `span`-Element angezeigt. Es gibt eine Reihe weiterer HTML-Elemente, in denen Sie Daten aus Datenfeldern externer Datenquellen anzeigen können. In anderen als diesen Elementen ist die Anzeige externer Daten nicht möglich. Folgende HTML-Elemente sind erlaubt:

### Verweise mit `<a>...</a>`

Angenommen, Sie haben eine externe Textdatei `verweis.txt` mit folgenden Zeilen:

```
URL,Verweistext
http://www.yahoo.de/,Yahoo Suchdienst
```

Diese Datei können Sie mit dem `object`-Element in HTML wie folgt einbinden:

```
<object id="Verweis" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
<param name="DataURL" value="verweis.txt">
<param name="UseHeader" value="true">
```

```
<param name="FieldDelim" value=",">
</object>
```

Danach können Sie folgenden Verweis konstruieren:

```
<a datasrc="#Verweis" datafld="URL"><span datafld="Verweistext"
datasrc="#Verweis"></span></a>
```

Indem Sie `<a datafld=>` notieren, wird der Inhalt, den das Datenfeld liefert, als Angabe für `href=` bewertet (im Beispiel liefert das Feld "URL" diese Angaben). Innerhalb des Verweises wird dann mit Hilfe des `span`-Elements das andere Feld »geholt«, nämlich der Verweistext. Solche Verweise machen beispielsweise Sinn, wenn Sie den gleichen Verweis an vielen verschiedenen Stellen auf Ihren Web-Seiten nutzen. Verweisziel und Verweistext sind dann zentral gespeichert und brauchen bei Änderungen nur in der kleinen Textdatei einmal geändert werden. Alle entsprechenden HTML-Verweise sind damit automatisch aktualisiert.

### Buttons mit `<button>...</button>`

Angenommen, Sie haben eine externe Textdatei *daten.txt* mit folgendem Aufbau und Beispiel-einträgen:

```
Zeile,Text
1,Das ist ein Test
2,mit einem Button
3,spannend - was?
```

Diese Datei können Sie mit dem `object`-Element in HTML wie folgt einbinden:

```
<object id="Daten" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
<param name="DataURL" value="daten.txt">
<param name="UseHeader" value="true">
<param name="FieldDelim" value=",">
</object>
```

Danach können Sie beispielsweise folgende Tabelle konstruieren:

```
<table "id=Anzeige" datasrc="#Daten" datapagesize="1"><tr><td>
<form action=""><button datafld="Text"
onClick="document.all.Anzeige.nextPage()"></button></form>
</td></tr></table>
```

Damit springt der Anwender bei jedem Klick auf den Button einen Datensatz weiter, und die Buttonbeschriftung nimmt den aktuellen Wert des Datenfeldes `Text` an.

### Bereiche mit `<div>...</div>`

Mit diesem HTML-Tag können Sie beispielsweise innerhalb einer Tabelle externe Daten ausgeben, genau so wie es im Beispiel *Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2) mit dem `span`-Element gemacht wird. Der Unterschied zwischen `<div>...</div>` und `<span>...</span>` ist, dass das `div`-Element einen eigenen Anzeigeblock erzeugt, das `span`-Element dagegen nicht.

**Grafiken mit <img>**

Angenommen, Sie wollen eine kleine Diashow programmieren. Dazu können Sie eine Textdatei *daten.txt* anlegen, die etwa folgenden Aufbau hat:

```
Nummer,Bild
1,meinhaus.gif
2,meinboot.gif
3,meinauto.gif
4,undich.gif
```

Diese Datei können Sie mit dem `object`-Element in HTML wie folgt einbinden:

```
<object id="Daten" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
<param name="DataURL" value="daten.txt">
<param name="UseHeader" value="true">
<param name="FieldDelim" value=",">
</object>
```

Danach können Sie beispielsweise folgende Tabelle konstruieren:

```
<table id="Anzeige" datasrc="#Daten" datapagesize="1"><tr>
<td>
Bild <span datafld="Nummer"></span><br> <input type="button" value=" < "
onclick="document.all.Anzeige.previousPage();">
<input type="button" value=" > " onclick="document.all.Anzeige.nextPage();">
</td><td>
<img datafld="Bild">
</td></tr></table>
```

Damit kann der Anwender mit den Blätter-Buttons die Bilder durchblättern.

**Frames mit <frame>**

Angenommen, Sie haben eine externe Textdatei *besteseite.txt* mit folgenden Zeilen:

```
BesteSeite
http://selfhtml.teamone.de/
```

Nun können Sie folgende HTML-Datei mit einem Frameset konstruieren:

```
<html><head>
<object id="UrlDerWoche" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
<param name="DataURL" value="besteseite.txt">
<param name="UseHeader" value="true">
</object></head>
<frameset rows="100,*">
<frame src="meine.htm">
<frame datasrc="#UrlDerWoche" datafld="BesteSeite">
</frameset>
</html>
```

Auf diese Weise können Sie in einem Frame-Fenster eine Seite anzeigen, deren Zieladresse Sie in der separaten Textdatei pflegen. Wichtig ist, dass Sie das `object`-Element genau an der Stelle platzieren wie im Beispiel.

### **Eingebettete Frames mit `<iframe>`**

Siehe das Beispiel bei Frames. Der Unterschied ist, dass Sie das `object`-Element in einer normalen HTML-Datei platzieren. Mit einem Befehl wie:

```
<iframe datasrc="#UrlDerWoche" datafld="BesteSeite"></iframe>
```

können Sie dann in der gleichen HTML-Datei in einer eingebetteten Fensterbox den Inhalt der Seite anzeigen, die in der Textdatei angegeben ist.

### **Labels mit `<label>...</label>`**

Das `label`-Element, das neu ist in HTML 4.0, erlaubt die Beschriftung eines Formularfeldes. Im Rahmen der Datenanbindung können Sie dieses HTML-Tag benutzen, wenn Sie die externen Daten in Formularfeldern ausgeben. Sie können dann dem `label`-Element, das Sie etwa einem Eingabefeld zuordnen, dynamisch den Feldnamen zuweisen, der in der externen Datendatei für dieses Feld definiert ist. Wenn sich der Feldname in der Datendatei dann ändert, wird er bei der Anzeige in der HTML-Datei automatisch aktualisiert. Beispiel:

```
<label for="Vorname" datafld="prenome"></label>
```

Dadurch wird dem Inhalt des `label`-Elements eines Formularfeldes der Feldname `prenome` zugewiesen, der in der Datenbank definiert ist.

### **Lauftexte mit `<marquee>...</marquee>`**

Mit diesem HTML-Element, das mittlerweile allerdings als Musterbeispiel proprietärer HTML-Erweiterungen verschrien ist, können Sie beispielsweise innerhalb einer Tabelle externe Daten ausgeben, genau so, wie es im Beispiel *Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2) mit dem `span`-Element gemacht wird. Der Unterschied zwischen `<marquee>...</marquee>` und `<span>...</span>` ist, dass das `marquee`-Element einen Lauftext erzeugt, das `span`-Element dagegen nicht.

### **Bereiche mit `<span>...</span>`**

Mit diesem HTML-Tag können Sie beispielsweise innerhalb einer Tabelle externe Daten ausgeben, genau so, wie es im Beispiel *Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2)

### **Formularfelder mit `<input>`**

Mit diesem HTML-Element können Sie beispielsweise innerhalb einer Tabelle externe Daten in editierbaren Formularfeldern ausgeben. Das ist besonders dann interessant, wenn geänderte Daten auch tatsächlich in die Datenquelle zurückgeschrieben werden können. Bei einfachen Textdateien ist das derzeit nicht möglich, wohl aber beim Datenmodell Remote Data Service (RDS) und beim Datenmodell JDBC DataSource Applet.

Die Einbindung in HTML erfolgt ähnlich wie im Beispiel *Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2), mit dem Unterschied, dass Sie die Regeln zum Einbinden des gewünschten Datenmodells beachten müssen. Notieren Sie im einleitenden `<table>`-Tag der Tabelle, in der Sie die Datensätze anzeigen, möglichst die Angabe `datapagesize="1"`. Innerhalb der Tabellenzeile, in der Sie externe Daten auf diese Weise anzeigen wollen, können Sie dann notieren (angenommene Beispiele):

```
<input type="text" datafld="Vorname">
```

oder:

```
<input type="radio" name="Kunde" value="Bereits Kunde" datafld="BereitsKunde">
```

oder:

```
<input type="checkbox" name="Zahlweise" value="Mastercard" datafld="Zahlungsweise">
```

Datenfelder, die in Eingabefeldern vom Typ `<input type="text">` angezeigt werden, können normale Texte sein, wie etwa ein Name. Radiobuttons und Checkboxes stellen dagegen andere Anforderungen. Ein Radiobutton wird bei der Anzeige dann aktiviert, wenn der aktuelle Feldinhalt in der Datenbank mit dem Wert übereinstimmt, der hinter der Angabe `value=` steht. Eine Checkbox wird dann markiert, wenn der betreffende Feldinhalt in der Datenbank 1, true oder einen beliebigen Inhalt mit mehr als einem Zeichen enthält. Wenn der Inhalt 0, false oder leer ist, wird die Checkbox nicht markiert.

### Formularfelder mit `<select>`

Dabei gelten die gleichen Bedingungen wie bei Formularfeldern vom Typ `input`. Wenn der Wert eines aktuellen Eintrags in der Datenbank mit einem der Werte übereinstimmt, die in der `value`-Angabe bei einem `option`-Element innerhalb der Auswahlliste definiert sind, ist diese Auswahl selektiert. Interessant ist diese Ausgabemöglichkeit von externen Daten wie bei anderen Formularfeldern nur, wenn auch ein Zurückschreiben geänderter Daten möglich ist. Bei Auswahllisten etwa kann ein neuer ausgewählter Wert als neuer Feldinhalt gespeichert werden. Der MS Internet Explorer unterstützt dabei nur die einfache Auswahl, Mehrfachauswahl nicht.

### Andere Objekte mit `<object>...</object>`

Mit dem `object`-Element können Sie genauso arbeiten wie mit dem `img`-Element. Der Unterschied ist, dass Sie beim `img`-Element auf Grafiken beschränkt sind, während Sie mit dem `object`-Element beliebige Multimedia-Dateien oder ActiveX-Controls einbinden können. So können Sie, analog zu dem Beispiel beim `img`-Element, anstelle einer »Diashow« eine »Musikshow« programmieren, die zum Beispiel aus MIDI-Dateien besteht.

In der externen Textdatei notieren Sie die gewünschten MIDI-Dateien, und bei der Einbindung in HTML können Sie dann anstelle der Datenanbindung mit `img` so etwas notieren wie:

```
<object datafld="Musik" width="0" height="0"></object>
```

(wobei »Musik« in der Textdatei mit den Daten als Feldüberschrift notiert sein muss).

### Java-Applets mit <applet>...</applet>

Sie können das Feature der Datenanbindung nutzen, um einen Parameter eines Java-Applets aus einer externen Datenquelle zu versorgen. Beispiel:

```
<applet code="verzerr.class">
  <param name="bild" datasrc="#aktuell" datafld="b_gates_bild">
</applet>
```

### 21.5.5 Datenmodell: Tabular Data Control (TDC)

Dies ist das Datenmodell für den Fall, dass die externe Datenquelle eine einfache Textdatei ist, die in jeder Zeile einen Datensatz enthält, und bei der die einzelnen Datenfelder der Datensätze durch ein Begrenzerzeichen, zum Beispiel ein Komma, markiert sind. Geeignet ist dieses Datenmodell für kleinere Datenbestände mit bis zu ein paar tausend Datensätzen. Ein vollständiges Beispiel dafür finden Sie weiter oben im Abschnitt *Beispiel Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2). Um eine solche Datenquelle an eine HTML-Datei anzubinden, müssen Sie folgendes Objekt in der HTML-Datei notieren.

Schema zur Einbindung in HTML:

```
<object id="Objektname" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name= value=>
  <!-- usw. weitere Parameter -->
</object>
```

Um solche Daten einzubinden, verwenden Sie das im Internet Explorer 4.x eingebaute ActiveX-Control mit der ID 333C7BC4-460F-11D0-BC04-0080C7055A83. Innerhalb des <object>-Tags notieren Sie verschiedene Parameter, die für die Datenpräsentation wichtig sind. Folgende Parameter sind erlaubt:

| Notation                            | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <param name="AppendData" value=>    | Mit diesem Parameter können Sie angeben, ob in der Ausgabetabelle beim Blättern für jedes neue Datensatz-Set (entsprechend der Größe von datapagesize) die vorhandenen Tabellenzellen überschrieben werden sollen oder ob neue Tabellenzellen angehängt werden sollen. Mit value=true erzwingen Sie das Anhängen von Daten. Per Voreinstellung werden die vorhandenen Tabellenzellen überschrieben. |
| <param name="CaseSensitive" value=> | Mit diesem Parameter können Sie angeben, ob bei Vergleichen von Anwendereingaben mit Daten aus der Datenquelle Groß-/Kleinschreibung unterschieden werden soll. Nur interessant bei der Weiterverarbeitung von Daten mit Scripts. Per Voreinstellung wird Groß-/Kleinschreibung unterschieden. Mit value=false können Sie die Fallunterscheidung ausschalten.                                       |



| Notation                                          | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;param name="CharSet" value=&gt;</pre>    | <p>Mit diesem Parameter können Sie den Zeichensatz angeben, der für die Daten in der Datendatei maßgeblich ist. Per Voreinstellung ist das der Zeichensatz Western Latin-1 (iso-8859-1). Verwenden Sie diesen Parameter nur, wenn die Daten einen anderen Zeichensatz benutzen, z.B. iso-8859-5 für Kyrillisch. Bei <code>value=</code> können Sie den Zeichensatz angeben. Nähere Informationen im Abschnitt 23.2 über Zeichensätze.</p>                                                                                                                                                                                                                                                                              |
| <pre>&lt;param name="DataURL" value=&gt;</pre>    | <p>Dieser Parameter ist unbedingt erforderlich. Bei <code>value=</code> geben Sie die Datei an, in der sich die Daten befinden.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <pre>&lt;param name="EscapeChar" value=&gt;</pre> | <p>Diesen Parameter sollten Sie angeben, wenn in Ihrer Datendatei ein Zeichen als Maskierungszeichen verwendet wird. Das ist z.B. der Fall, wenn Sie in einer kommagegrenzten Datei Kommata innerhalb von Feldern mit einem vorangestellten Backslash \ maskieren. Geben Sie in diesem Fall bei <code>value=</code> das Maskierungszeichen an.</p>                                                                                                                                                                                                                                                                                                                                                                     |
| <pre>&lt;param name="FieldDelim" value=&gt;</pre> | <p>Mit diesem Parameter geben Sie an, welches Zeichen als Trennzeichen für Datenfelder innerhalb einer Zeile der externen Textdatei interpretiert werden soll. Bei <code>value=</code> können Sie das Zeichen angeben.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <pre>&lt;param name="Filter" value=&gt;</pre>     | <p>Mit diesem Parameter können Sie die Datenauswahl einschränken. Per Voreinstellung werden alle Daten aus der externen Datei angezeigt. Interessant ist dieser Parameter vor allem, wenn der Anwender in einem Eingabeformular die Suche selbst definieren kann und die Eingaben dann mit Hilfe eines Scripts dynamisch diesem Parameter des <code>&lt;object&gt;</code>-Tags zugewiesen werden. Bei <code>value=</code> können Sie den Filter angeben. Beispiele:<br/> <code>value="Telefon &lt; '089*'"</code> findet in einem Feld »Telefon« nur Nummern, die mit 089 beginnen.<br/> <code>value="PLZ &gt; '50000'"</code> findet in einem Feld »PLZ« für Postleitzahlen nur Werte, die größer als 50000 sind.</p> |
| <pre>&lt;param name="Language" value=&gt;</pre>   | <p>Mit diesem Parameter können Sie die Sprache angeben, in der die Daten der externen Datendatei vorliegen. Maßgeblich sind dafür die Sprachennamen, wie sie in ISO 369 definiert sind. Bei <code>value=</code> können Sie die Sprache angeben.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <pre>&lt;param name="RowDelim" value=&gt;</pre>   | <p>Wenn Ihre externe Datendatei so aufgebaut ist, dass am Ende eines Datensatzes ein Datensatzbegrenzerzeichen steht, müssen Sie dieses Zeichen mit diesem Parameter angeben. Notieren Sie hinter <code>value=</code> das Zeichen.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <pre>&lt;param name="Sort" value=&gt;</pre>       | <p>Mit diesem Parameter können Sie veranlassen, dass bei gleichzeitiger Ausgabe mehrerer Datensätze (also bei <code>datapagesize=</code> größer 1) die angezeigten Datensätze nach einem bestimmten Feld sortiert werden. Beispiele:<br/> <code>value="PLZ"</code> sortiert auszugebende Daten nach einem Feld namens »PLZ«.<br/> <code>value="-PLZ"</code> sortiert auszugebende Daten nach einem Feld namens »PLZ«, und zwar absteigend (Minuszeichen davor).<br/> <code>value="Name; PLZ"</code> sortiert auszugebende Daten erstens nach Name und zweitens nach Postleitzahlen.</p>                                                                                                                                |

| Notation                                             | Bedeutung                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;param name="TextQualifier" value=&gt;</pre> | <p>Wenn Ihre externe Datendatei so aufgebaut ist, dass alle Datenfelder in bestimmte Zeichen eingeschlossen sind, zum Beispiel in Anführungszeichen, dann sollten Sie mit diesem Parameter das betreffende Zeichen angeben. Mit <code>value="'"</code> etwa bestimmen Sie das Anführungszeichen als Zeichen für Feldanfang und Feldende.</p>                                    |
| <pre>&lt;param name="UseHeader" value=&gt;</pre>     | <p>Diesen Parameter müssen Sie dann angeben, wenn die erste Zeile der externen Datendatei keine Daten enthält, sondern die zugehörigen Feldnamen. Mit <code>value="true"</code> wird die Datendatei so interpretiert, dass die erste Zeile Feldnamen enthält. Mit <code>value="false"</code> wird sie so interpretiert, dass gleich in der ersten Zeile die Daten beginnen.</p> |

### 21.5.6 Datenmodell: Remote Data Service (RDS)

Über das Datenmodell Remote Data Service (RDS) können Sie die Anbindung einer HTML-Datei an Daten aus einer Datenbank realisieren. Geeignet ist dieses Modell für Datenbanken, die über die Datenbankschnittstellen OLE-DB oder ODBC (DBMS) verfügen. Das sind beispielsweise MS Access, Oracle oder SQL-Server wie `msql`. Über die entsprechenden Produkte und den Aufbau von Datenbanken damit informieren Sie sich bitte bei den betreffenden Herstellern. Um eine solche Datenanbindung im WWW zu realisieren, muss die Datenbankanwendung auf dem Server-Rechner installiert sein. Fragen Sie dazu gegebenenfalls Ihren Provider. Um eine solche Datenquelle an eine HTML-Datei anzubinden, müssen Sie folgendes Objekt in der HTML-Datei notieren.

Schema zur Einbindung in HTML:

```
<object id="Objektnamen" classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33">
  <param name= value=>
  <!-- usw. weitere Parameter -->
</object>
```

Um solche Daten einzubinden, verwenden Sie das im Internet Explorer 4.x eingebaute ActiveX-Control mit der ID `BD96C556-65A3-11D0-983A-00C04FC29E33`. Innerhalb des `<object>`-Tags notieren Sie verschiedene Parameter, die für die Datenpräsentation wichtig sind. Folgende Parameter sind möglich:

| Notation                                           | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;param name= "AppendData" value=&gt;</pre> | <p>Mit diesem Parameter können Sie angeben, ob in der Ausgabetable beim Blättern für jedes neue Datensatz-Set (entsprechend der Größe von <code>datapagesize</code>) die vorhandenen Tabellenzellen überschrieben werden sollen oder ob neue Tabellenzellen angehängt werden sollen. Mit <code>value=true</code> erzwingen Sie das Anhängen von Daten. Per Voreinstellung werden die vorhandenen Tabellenzellen überschrieben.</p> |

| Notation                                               | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;param name="CaseSensitive" value=&gt;</code> | Mit diesem Parameter können Sie angeben, ob bei Vergleichen von Anwendereingaben mit Daten aus der Datenquelle Groß-/Kleinschreibung unterschieden werden soll. Nur interessant bei der Weiterverarbeitung von Daten mit Scripts. Per Voreinstellung wird Groß-/Kleinschreibung unterschieden. Mit <code>value=false</code> können Sie die Fallunterscheidung ausschalten.                                                         |
| <code>&lt;param name="CharSet" value=&gt;</code>       | Mit diesem Parameter können Sie den Zeichensatz angeben, der für die Daten in der Datendatei maßgeblich ist. Per Voreinstellung ist das der Zeichensatz Western Latin-1 (iso-8859-1). Verwenden Sie diesen Parameter nur, wenn die Daten einen anderen Zeichensatz benutzen, z.B. iso-8859-5 für Kyrillisch. Bei <code>value=</code> können Sie den Zeichensatz angeben. Nähere Informationen im Abschnitt 23.2 über Zeichensätze. |

**Beachten Sie:** Solche Datenanbindungen sind nur sinnvoll, wenn Sie dem Anwender ein HTML-Formular zur Verfügung stellen, in dem er seine Angaben zum Durchsuchen der Datenbank machen kann. Mit Hilfe eines JavaScripts können Sie dann die Anwendereingaben in einen gültigen SQL-Befehl umwandeln, den Sie im Script dem Parameter `SQL` des `object`-Tags zuweisen. So können Sie beispielsweise mit der JavaScript-Anweisung:

```
document.all.Datenbank.SQL = Abfrage;
```

eine SQL-Abfrage zuordnen. In diesem Beispiel ist "Datenbank" der Name, der beim `object`-Tag als `id=` vergeben wird, und `Abfrage` ist eine Variable, die zuvor aus Formulareingaben des Anwenders einen SQL-Befehl kreiert hat. Bei solchen Datenbanken ist es auch möglich, SQL-Befehle zum Schreiben von geänderten Datensätzen in die Datenbank zurückzuschreiben. Voraussetzung ist, dass die Daten in der HTML-Datei in editierbaren HTML-Eingabefeldern (z.B. `<input...>`) angezeigt werden.

Ausführliche Informationen zum Einrichten und Realisieren einer Datenbank mit Scripts finden Sie auf den Entwicklerseiten von Microsoft ([msdn.microsoft.com/](http://msdn.microsoft.com/)). Suchen Sie dort nach *Microsoft Internet Client SDK*.

## 21.5.7 Datenmodell: JDBC DataSource Applet

Das Datenmodell JDBC DataSource Applet greift ebenso wie das Datenmodell Remote Data Service (RDS) auf Datenbanken zu, die über die Datenbankschnittstellen OLE-DB oder ODBC (DBMS) verfügen, zum Beispiel MS Access, Oracle oder SQL-Server wie `mysql`. Der Unterschied zwischen dem JDBC DataSource Applet und dem Remote Data Service (RDS) ist, dass Letzterer in der HTML-Datei als ActiveX-Control eingebunden wird, während das JDBC DataSource Applet ein Java-Applet ist. Damit will Microsoft seine Datenanbindungstechnologie anderen Browsern zur Verfügung stellen, die kein ActiveX unterstützen, wohl aber Java. Das tun zwar Netscape und einige andere Browser, doch solange diese das Konzept der Datenanbindung mit Dynamischem HTML nicht unterstützen, macht das Ganze eigentlich wenig Sinn.

Das Java-Applet, das diese Form der Datenanbindung steuert, gehört nicht zum Lieferumfang des MS Internet Explorers. Auf den Web-Seiten von Microsoft ([www.microsoft.com/](http://www.microsoft.com/)) können Sie das JDBC DataSource Applet downloaden. Suchen Sie dort auf den Seiten nach *JDBC Data-*

*Source Applet.* Um eine solche Datenanbindung im WWW zu realisieren, muss die Datenbank-anwendung auf dem Server-Rechner installiert sein. Fragen Sie dazu gegebenenfalls Ihren Provi-der. Um eine solche Datenquelle an eine HTML-Datei anzubinden, müssen Sie das JDBC Data-Source Java-Applet in der HTML-Datei notieren.

Schema zur Einbindung in HTML:

```
<applet code="JDC.class" name="DSA1" id="DSA1" width="0" height="0">
  <param name= value=>
  <!-- usw. weitere Parameter -->
</applet>
```

Das Java-Applet wird wie in HTML üblich mit dem `<applet>`-Tag eingebunden. Mit dem Attri-but `code=` geben Sie die ausführbare Java-Datei an. Dies ist die Datei *JDC.class*, die Sie mit dem Download des JDBC DataSource Java-Applets erhalten. Innerhalb des Applets notieren Sie verschiedene Parameter, die das Applet benötigt, um korrekt zu arbeiten. Weiterführende Infos zu den Parametern entnehmen Sie der Originaldokumentation zu dem Applet. Folgende Para-meter erwartet das Applet:

| Notation                                              | Bedeutung                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;param name="cabbase" value=&gt;</code>      | Mit diesem Parameter geben Sie die Datei an, in der der Quellcode des Applets steht. Es handelt sich um eine CAB-Datei. Relative Pfadnamen oder auch absolute Adressen sind dabei erlaubt.                                                                                                                                                             |
| <code>&lt;param name="dbURL" value=&gt;</code>        | Mit diesem Parameter geben Sie bei <code>value=</code> die Datei an, in der die Datenbank gespeichert ist. Relative Pfadnamen oder auch absolute Adressen sind dabei erlaubt.                                                                                                                                                                          |
| <code>&lt;param name="showUI" value=&gt;</code>       | Mit diesem Parameter geben Sie an, ob das Applet innerhalb der HTML-Datei seine eigene Benutzeroberfläche anzeigen soll oder nicht. Mit <code>value="true"</code> wird das Applet angezeigt, mit <code>value="false"</code> nicht. Der Parameter ist optional.                                                                                         |
| <code>&lt;param name="sqlStatement" value=&gt;</code> | Mit diesem Parameter müssen Sie einen Befehl angeben, den Sie zum Ansprechen der Datenbank absetzen wollen. Dies kann ein beliebiger gültiger SQL-Befehl sein. SQL (Structured Query Language) ist eine standardisierte Datenbankabfragesprache. Diese Sprache und ihre Befehle müssen Sie kennen, um den Parameter mit korrekten Werten zu versorgen. |
| <code>&lt;param name="allowInsert" value=&gt;</code>  | Mit diesem Parameter geben Sie an, ob der Anwender Datensätze durch Ausfüllen entsprechender Eingabefelder neue Datensätze in die Datenbank schreiben darf. Mit <code>value="true"</code> erlauben Sie das Hinzufügen von Daten, mit <code>value="false"</code> verbieten sie es.                                                                      |
| <code>&lt;param name="allowDelete" value=&gt;</code>  | Mit diesem Parameter geben Sie an, ob der Anwender angezeigte Datensätze mit Hilfe entsprechender Befehls-Buttons aus der Datenbank löschen darf. Mit <code>value="true"</code> erlauben Sie das Löschen von Daten, mit <code>value="false"</code> verbieten sie es.                                                                                   |

| Notation                                           | Bedeutung                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;param name="allowUpdate" value=&gt;</pre> | <p>Mit diesem Parameter geben Sie an, ob der Anwender angezeigte Datensätze in entsprechenden Eingabefeldern editieren und geänderte Daten mit Hilfe entsprechender Befehls-Buttons in die Datenbank zurückschreiben darf. Mit <code>value="true"</code> erlauben Sie das Ändern von Daten, mit <code>value="false"</code> verbieten sie es.</p> |
| <pre>&lt;param name="user" value=&gt;</pre>        | <p>Mit diesem Parameter können Sie der Datenbank bei <code>value=</code> den Namen eines angemeldeten Users übergeben, sofern die Datenbank dies verlangt. Der Parameter ist optional.</p>                                                                                                                                                       |
| <pre>&lt;param name="password" value=&gt;</pre>    | <p>Mit diesem Parameter können Sie der Datenbank bei <code>value=</code> das Passwort des Users übergeben, sofern die Datenbank dies verlangt. Der Parameter ist optional.</p>                                                                                                                                                                   |

**Beachten Sie:** Beim Datenmodell JDBC DataSource Applet gelten die Anmerkungen zum Datenmodell Remote Data Service (RDS) in Kapitel 21.5.6.

### 21.5.8 Datenmodell: XML Data Source

Das Datenmodell XML Data Source könnte in Zukunft sehr interessant werden, da es die vom W3-Konsortium standardisierte Sprache XML zur Definition von Daten benutzt. Bei diesem Datenmodell werden die Daten in einer XML-Datei gespeichert. Für Datensätze und Datenfelder können Sie innerhalb der XML-Datei eigene Tags und deren Verschachtelung erfinden. Ein denkbare Schema für eine Adressendatei wäre etwa:

```
<adresse>
  <name>Meyer</name>
  <vorname>Hubert</vorname>
  <ort>Hamburg</ort>
  <telefon>040-598487374</telefon>
</adresse>
```

Im Lieferumfang des MS Internet Explorers 4.x ist ein Java-Applet enthalten, das solche Tag-Strukturen aus XML-Dateien interpretiert und die Datenstruktur daraus ermittelt. Um diese Art von Datenquelle an eine HTML-Datei anzubinden, müssen Sie das zugehörige Java-Applet in der HTML-Datei notieren.

Schema zur Einbindung in HTML:

```
<applet class=com.ms.xml.dso.xmlldso.class id="xmlldso" width="0" height="0"
mayscript=true>
  <param name="url" value=>
</applet>
```

Das Java-Applet wird wie in HTML üblich mit dem `<applet>`-Tag eingebunden. Notieren Sie das einleitende `<applet>`-Tag genau so wie im obigen Schema. Der Grund für die etwas unge-

wöhnliche Angabe bei `class=` ist, dass dieses Java-Applet eine interne Ressource des MS Internet Explorers ist und keine separat vorliegende Datei.

Das Applet erwartet genau einen Parameter:

- Mit `<param name="url" value=>` geben Sie bei `value=` den Dateinamen der XML-Datei an. Dabei sind natürlich auch relative Pfadangaben oder absolute Adressen erlaubt.

## 21.5.9 Datenmodell: MSHTML Data Source

Was Sie mit XML besonders elegant lösen können, geht etwas weniger elegant auch mit HTML selbst: Hinter dem Datenmodell MSHTML Data Source verbirgt sich nichts anderes als eine Konvention, nach der die Daten in einer HTML-Datei gehalten werden – diese HTML-Datei dient dann als externe Quelle für eine andere HTML-Datei, in der die Daten angezeigt werden. Ein Schema für den Datensatz einer Adressendatei wäre etwa:

```
<div id="Adresse">
  <span id="Name">Meyer</span><br>
  <span id="Vorname">Hubert</span><br>
  <span id="Ort">Hamburg</span><br>
  <span id="Telefon">040-598487374</span><br>
</div>
```

Um diese Art von Datenquelle an eine HTML-Datei anzubinden, müssen Sie ein einfaches Objekt dafür notieren.

Schema zur Einbindung in HTML:

```
<object id="Daten" data="daten.htm" width="0" height="0">
</object>
```

Innerhalb des `<object>`-Tags vergeben Sie mit `id=` wie üblich einen Namen, unter dem Sie das Objekt ansprechen und die Daten aus der anderen Datei in einer Tabelle dynamisch anzeigen können. Das Zugreifen auf die Datensätze funktioniert dabei genau so wie im Beispiel *Datei dynamisch in Tabelle einbinden* (Kapitel 21.5.2). Bei `datafld=` geben Sie den Namen an, den Sie in der externen HTML-Datei bei `id=` für jedes gleiche Feld eines Datensatzes vergeben haben, z.B. "Vorname".

## 21.6 Dynamische Filter (nur Microsoft)

### 21.6.1 Allgemeines zu dynamischen Filtern

Der MS Internet Explorer unterstützt ab der Version 4.x spezielle CSS-Eigenschaften, die so genannten Filter. Mit Hilfe solcher Filter sind grafische Effekte möglich, wie sie aus professionellen Grafikprogrammen bekannt sind: Schatteneffekte für Texte oder auch Transparenzdefinitionen für Grafiken, die auf einer WWW-Seite angezeigt werden, gehören zum Repertoire der möglichen Angaben. Als reine CSS-Eigenschaften sind diese Filter statisch. Mit Hilfe von Scripts können Sie die entsprechenden Angaben zu den Filtern jedoch dynamisch beeinflussen. Ferner gibt es zwei Spezialfilter für Blend- und diverse Übergangseffekte. Mit etwas Programmierkenntnis-

sen und ästhetischem Gefühl können dabei so ziemlich alle Träume wahr werden, die Web-Designer heimlich immer schon geträumt haben: Grafiken, die sich wie von Geisterhand langsam ein- und ausblenden, fließende Übergänge zwischen Elementen, Texte, die verschwimmen usw.

Der Script-Zugriff auf definierte Filter erfolgt über das `filter`-Objekt in JScript/JavaScript. Die Beispiele hier können nicht alle Möglichkeiten des `filter`-Objekts behandeln. Sie dienen nur zur Veranschaulichung, wie Filter sich durch ein Script dynamisch verändern lassen. Die Filter-Technik von Microsoft ist jedoch nie ein Standard geworden, und es ist fraglich, ob sie je zum Standard gehören werden.

### 21.6.2 Einfaches Dynamisches Ändern von Filterwerten

Durch zeitgesteuertes Ändern der CSS-Filtereigenschaften können Sie interessante Effekte erzielen. Ein Beispiel:

```
<html><head><title>Test</title>
</head><body>
<div id="Zittertext" style="width:100%; font-size:72pt; font-weight:bold;
color:#FF6666;
filter:Wave(freq=5, light=0, phase=80, strength=1);">Leicht gest&ouml;rt</div>
<script language="JScript" type="text/jscript">
<!--
function DynWave() {
    if(document.all.Zittertext.filters[0].freq > 30)
        document.all.Zittertext.filters[0].freq = 5;
    document.all.Zittertext.filters[0].freq += 1;
    if(document.all.Zittertext.filters[0].phase > 100)
        document.all.Zittertext.filters[0].phase = 0;
    document.all.Zittertext.filters[0].phase += 10;
    if(document.all.Zittertext.filters[0].strength > 10)
        document.all.Zittertext.filters[0].strength = 1;
    document.all.Zittertext.filters[0].strength += 1;
    window.setTimeout("DynWave()",100);
}
DynWave();
//-->
</script>
</body>
</html>
```

Im Beispiel wird in einem `<div>`-Bereich der zugehörige Text (`leicht gest&ouml;rt`) mit Hilfe von CSS-Eigenschaften formatiert. Zu den Formatierungen gehört auch die Anwendung des Filters `Wave()`. Damit wird der Effekt des Filters zwar bereits angezeigt, aber es bewegt sich noch nichts. In einem Script unterhalb des so definierten `<div>`-Bereichs werden die Angaben dieses Filters dynamisch verändert. Da ein Endlos-Effekt mit immer neuen Werten und Kombinationen eingebaut ist, entsteht ein Zitter-Effekt.



21.20: Der Filter lässt die Buchstaben erzittern.

Die JavaScript-Funktion wird im obigen Beispiel unterhalb des Bereichs, auf den sie sich bezieht, definiert. Der Grund ist, dass die JavaScript-Anweisungen sofort ausgeführt werden. Der `<div>`-Bereich, auf dessen Daten die Anweisungen dynamisch zugreifen, sollte zu diesem Zeitpunkt bereits eingelesen und dem Browser bekannt sein. Deshalb steht das Script unterhalb des `<div>`-Bereichs. Andernfalls könnte es zu Fehlermeldungen kommen.

Innerhalb des Script-Bereichs wird zunächst eine Funktion mit dem Namen `DynWave()` definiert. Diese Funktion ändert die Parameter `freq`, `phase` und `strength` des Wave-Filters, der bei dem `<div>`-Bereich definiert wurde. Am Ende ruft sich die Funktion selbst wieder auf. Dadurch entsteht ein Endlos-Effekt. Der Aufruf erfolgt zeitverzögert um 100 Millisekunden. Das ist wichtig, da der Selbstaufwurf der Funktion sonst »unendlich schnell« wäre und ein Problem im Arbeitsspeicher verursachen würde (Selbstaufrufe von Funktionen sind nämlich nicht ganz unkritisch, da für jeden Funktionsaufruf neuer Arbeitsspeicherplatz reserviert werden muss). Da die Funktion zu Beginn überhaupt erst einmal aufgerufen werden muss (sonst würde gar nichts passieren), steht am Ende des Script-Bereichs – außerhalb der Funktion `DynWave()` – der einmalige Funktionsaufruf `DynWave()`.

Innerhalb der Funktion im Beispiel können Sie erkennen, wie auf Filter dynamisch zugegriffen wird. Der Zugriff geschieht mit Hilfe des `all`-Objekts. Bezug ist die Angabe `id="Zittertext"`, die im `<div>`-Tag vergeben wird. Mit `document.all.Zittertext` ist dann der Zugriff auf dieses HTML-Element möglich.

Zwei Besonderheiten sind bei Filtern jedoch zu beachten:

- Mit einem Ausdruck wie `document.all.Zittertext.filters[0].strength` greifen Sie auf den Wert zu, der beim ersten definierten Filter des HTML-Elements bei der Angabe `strength=` als Parameter notiert ist. Da die Filter zu den Stylesheet-Angaben gehören, müsste eigentlich `document.all.Zittertext.style.filters[0].strength` notiert werden. Dies führt beim MS Internet Explorer jedoch zu Fehlermeldungen. Lassen Sie die Angabe `style` in der Mitte also weg.
- Da ein Element mehrere Filter haben kann, müssen Sie genau angeben, welchen Filter Sie dynamisch ansprechen wollen. Im obigen Beispiel wird mit `filters[0]` der erste definierte Filter angesprochen. Falls im gleichen `<div>`-Tag noch ein zweiter Filter definiert wäre,



könnten Sie diesen mit `filters[1]` ansprechen, also einfach bei 0 zu zählen beginnen. Microsoft erlaubt auch weitere Notationsmöglichkeiten, für das obige Beispiel etwa `document.all.Zittertext.filters["wave"].strength` oder `document.all.Zittertext.filters.wave.strength`.

Das hier beschriebene Schema zum Zugreifen auf Filterwerte können Sie auf alle anderen Filter ebenso anwenden. Auch auf solche, die vor allem in Verbindung mit Grafiken sinnvoll sind.

### 21.6.3 Fließende Übergänge mit dem blendTrans-Filter

Der `blendTrans`-Filter ist ein spezieller Filter, der nur in Verbindung mit Scripts sinnvoll ist. Er erlaubt es, einen fließenden Übergang von einer Grafik zu einer anderen Grafik oder von einem Textinhalt zu einem anderen Textinhalt zu definieren. Dadurch werden Effekte wie bei professionellen Diashows oder wie bei Fernsehbildern auf WWW-Seiten möglich. Ein Beispiel für Grafiken:

```
<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
Bild1 = new Image();
Bild1.src = "madrid1.jpg";
Bild2 = new Image();
Bild2.src = "madrid2.jpg";
var Bild = 1;
function Bildwechsel() {
  if (Bild == 1) {
    Bild = 2;
    document.all.Madrid.filters.blendTrans.Apply();
    document.all.Madrid.src = Bild2.src;
    document.all.Madrid.filters.blendTrans.Play();
  }
  else {
    Bild = 1;
    document.all.Madrid.filters.blendTrans.Apply();
    document.all.Madrid.src = Bild1.src;
    document.all.Madrid.filters.blendTrans.Play();
  }
}
//-->
</script>
</head><body>
<p>Klicken Sie auf das Bild, nachdem es vollständig angezeigt ist.</p>

</body>
</html>
```



21.21: Der Filter blendet sanft von einem Bild zu nächsten.

Das Beispiel bewirkt, dass beim Klicken auf ein Bild dieses Bild langsam in ein anderes übergeht. Wird das andere Bild dann angeklickt, geht es langsam wieder in das erste über. Dazu wird zunächst bei der Definition des Bildes mit `<img...>` eine spezielle Stylesheet-Angabe notiert: der Filter `blendTrans(...)`. Dieser Filter erwartet einen Parameter, nämlich die Angabe zu `Duration=`. Damit können Sie angeben, wie lange der Übergang von dem Bild zu seinem Nachfolger dauern soll. Im Beispiel wird 4 angegeben – das steht für 4 Sekunden. Sie können auch Bruchwerte bis zu tausendstel Sekunden angeben, etwa 2.450 (Dezimalzeichen ist ein Punkt!).

Damit ist der Übergangseffekt jedoch nur »registriert«. Um ihn tatsächlich auszuführen, ist ein Script erforderlich. Im obigen Beispiel wird im Dateikopf ein JavaScript-Bereich definiert. Dort werden zunächst mit Hilfe des Bildobjekts `image` beide betroffenen Bilder als Objekte (`Bild1` und `Bild2`) definiert. Der Eigenschaft `src` der Bildobjekte werden die gewünschten Grafikdateien zugewiesen. Diese Vorarbeit mit dem `image`-Objekt ist zwar nicht zwingend notwendig, hat aber den Vorteil, dass alle betroffenen Bilder bereits in den Arbeitsspeicher geladen sind, wenn der Übergang angestoßen wird.

Innerhalb der Funktion `Bildwechsel()` findet dann der eigentliche Übergang statt. Diese Funktion wird aufgerufen, wenn der Anwender auf das Bild klickt. Dazu ist im `<img>`-Tag der Event-Handler `onClick=` notiert.

In der Funktion `Bildwechsel()` wird zunächst die Methode `Apply()` des `blendTrans`-Filters aufgerufen. Dieser Aufruf ist notwendig, um dasjenige Objekt zu identifizieren, das in ein anderes übergehen soll. Im Beispiel ist das die Grafik, angesprochen über das `all`-Objekt und den vergebenen `id`-Namen `DynBild`. Als nächster Befehl wird der Grafik diejenige neue Grafik zugeordnet, durch die sie ersetzt werden soll. Erst dann sind alle Voraussetzungen erfüllt, um den eigentlichen Übergang zu starten. Dazu wird die Methode `Play()` des `filter`-Objekts benutzt.

Die Variable `Bild` und der `else`-Zweig in der Beispielfunktion dienen dazu, den Wechseleffekt sicherzustellen. Mit den beschriebenen Befehlen findet der Übergang ansonsten genau einmal statt. Der Übergangseffekt ist nicht nur bei Grafiken möglich, sondern auch bei Text. Ein Beispiel für Text:

```

<html><head><title>Test</title>
<script language="JScript" type="text/jscript">
<!--
var NormalText = "Wer HTML nicht ehrt";
var AndererText = "ist des Dynamischen nicht wert";
var Text = 1;
function Wechsel() {
  if (Text == 1) {
    Text = 2;
    document.all.MeinText.filters.blendTrans.Apply();
    document.all.MeinText.innerText = AndererText;
    document.all.MeinText.filters.blendTrans.Play();
  }
  else {
    Text = 1;
    document.all.MeinText.filters.blendTrans.Apply();
    document.all.MeinText.innerText = NormalText;
    document.all.MeinText.filters.blendTrans.Play();
  }
}
//-->
</script>
</head><body>
<p>Klicken Sie auf den folgenden Text:</p>

<div id="MeinText" style="cursor:hand; font-size:32pt; color:red; width:600;
height:200; filter:blendTrans(Duration=0.5)" onClick="Wechsel()">
Wer HTML nicht ehrt</div>
</body>
</html>

```

Anwendbar ist der `blendTrans`-Filter bei Text auf die HTML-Elemente

- `<body>...</body>`
- `<div>...</div>`
- `<span>...</span>`
- `<input>`
- `<button>`
- `<textarea>...</textarea>`
- `<marquee>...</marquee>`

sowie auf alle Tabellenelemente. Bei `<div>...</div>` und bei `<span>...</span>` müssen Sie Stylesheet-Angaben zu Breite und/oder Höhe des Elements notieren, sonst funktioniert es nicht.

Das obige Beispiel ist im Wesentlichen das gleiche wie bei den Grafiken. Im Unterschied dazu wird jedoch ein Übergang zwischen zwei Texten definiert. Der erste Text ist im Dateikörper der HTML-Datei mit `<div>...</div>` entsprechend den Regeln definiert. Die Realisierung des

Übergangs zwischen den beiden Texten ist ähnlich wie bei Grafiken. Anstelle der Objekteigenschaft `src` (bei Grafiken) müssen Sie bei Texten jedoch die Eigenschaft `innerText` (wie im Beispiel) oder `innerHTML` verwenden. Diese Eigenschaften erlauben das dynamische Austauschen des Inhalts, den das angesprochene HTML-Element enthält.

### 21.6.4 Fließende Übergänge mit dem `revealTrans`-Filter

Der `revealTrans`-Filter ist ganz ähnlich zu handhaben wie der `blendTrans`-Filter. Während der `blendTrans`-Filter jedoch nur einfache Blenden von einem Inhalt zum anderen erlaubt, stellt der `revealTrans`-Filter ein ganzes Arsenal grafischer Überblend-Effekte bereit. Ein Beispiel (für Grafik):

```
<html><head><title>Test</title>
</head><body>

<script language="JScript" type="text/jscript">
<!--
window.setTimeout("Wechsel()",5000);
function Wechsel()
{
document.all.DynBild.filters.revealTrans.Apply();
document.all.DynBild.src = "sun.gif";
document.all.DynBild.filters.revealTrans.Play();
}
//-->
</script>
</body></html>
```

Das Beispiel bewirkt, dass innerhalb von 5 Sekunden (5000 Millisekunden) das angezeigte Bild »rain.gif« zerbröselt und durch das Bild "sun.gif" ersetzt wird.

Beispiel (für Text):

```
<html><head><title>Test</title>
</head><body>
<div id="MeinText" style="width:600px; height:100px; font-size:24pt;
filter:revealTrans(Duration=1,Transition=7)">
Wer HTML nicht ehrt
</div>
<script language="JScript" type="text/jscript">
<!--
window.setTimeout("Wechsel()",2500);
function Wechsel()
{
document.all.MeinText.filters.revealTrans.Apply();
document.all.MeinText.innerText = "ist des Dynamischen nicht wert";
document.all.MeinText.filters.revealTrans.Play();
}
```

```

}
//-->
</script>
</body></html>

```

Das Beispiel bewirkt, dass der Text *Wer HTML nicht ehrt* nach 2,5 Sekunden durch den Text *ist des Dynamischen nicht wert* ersetzt wird, und zwar so, dass der neue Text den zunächst angezeigten Text von rechts her »überrollt«. Dazu wird zunächst der Text *Wer HTML nicht ehrt* innerhalb eines `<div>`-Bereichs notiert. Denn der `revealTrans`-Filter ist im Zusammenhang mit Text auf die folgenden HTML-Elemente anwendbar:

- `<body>...</body>`
- `<div>...</div>`
- `<span>...</span>`
- `<input>`
- `<button>`
- `<textarea>...</textarea>`
- `<marquee>...</marquee>`

sowie auf alle Tabellenelemente. Bei `<div>...</div>` und bei `<span>...</span>` müssen Sie Stylesheet-Angaben zu Breite und/oder Höhe des Elements notieren, sonst funktioniert es nicht. Im obigen Beispiel werden Breite und Höhe des `<div>`-Bereichs mit `width:` und `height:` festgelegt.

Ferner wird eine spezielle Stylesheet-Angabe notiert: der Filter `revealTrans(...)`. Dieser Filter erwartet zwei Parameter, nämlich die Angabe zu `Duration=` und eine Angabe zu `Transition=`. Mit `Duration=` können Sie angeben, wie lange der Übergang von dem Text zu seinem Nachfolger dauern soll. Im Beispiel wird 1 angegeben – das steht für eine Sekunde. Sie können auch Bruchwerte bis zu tausendstel Sekunden angeben, etwa 2.450 (Dezimalzeichen ist ein Punkt!). Bei `Transition=` geben Sie die Art an, wie der Filter wirken soll. Die erlaubten Werte beim `revealTrans`-Filter listet die folgende Tabelle auf.

| Notation                  | Bedeutung                              |
|---------------------------|----------------------------------------|
| <code>Transition=0</code> | Box-Effekt: außen nach innen           |
| <code>Transition=1</code> | Box-Effekt: innen nach außen           |
| <code>Transition=2</code> | Kreis-Effekt: von außen nach innen     |
| <code>Transition=3</code> | Kreis-Effekt: von innen nach außen     |
| <code>Transition=4</code> | Aufroll-Effekt: von unten nach oben    |
| <code>Transition=5</code> | Aufroll-Effekt: von oben nach unten    |
| <code>Transition=6</code> | Aufroll-Effekt: von links nach rechts  |
| <code>Transition=7</code> | Aufroll-Effekt: von rechts nach links  |
| <code>Transition=8</code> | Streifen-Effekt: von links nach rechts |
| <code>Transition=9</code> | Streifen-Effekt: von oben nach unten   |

| Notation      | Bedeutung                                        |
|---------------|--------------------------------------------------|
| Transition=10 | Rechteck-Effekt: von links nach rechts           |
| Transition=11 | Rechteck-Effekt: von oben nach unten             |
| Transition=12 | Zerbröselungseffekt: in alle Richtungen          |
| Transition=13 | Aufroll-Effekt: beidseitig von außen nach innen  |
| Transition=14 | Aufroll-Effekt: beidseitig von innen nach außen  |
| Transition=15 | Zuklapp-Effekt: von oben und unten               |
| Transition=16 | Aufklapp-Effekt: nach oben und unten             |
| Transition=17 | Aufroll-Effekt: von rechts oben nach links unten |
| Transition=18 | Aufroll-Effekt: von rechts unten nach links oben |
| Transition=19 | Aufroll-Effekt: von links oben nach rechts unten |
| Transition=20 | Aufroll-Effekt: von links unten nach rechts oben |
| Transition=21 | Jalousie-Effekt: waagrecht                       |
| Transition=22 | Jalousie-Effekt: senkrecht                       |
| Transition=23 | Zufallseffekt: aus Effekten 0-22                 |

### 21.6.5 Ein-/Ausblend-Effekte

Sie können die beiden Filter `blendTrans` und `revealTrans` auch dazu nutzen, um Texte oder Grafiken »aus dem Nichts heraus« einzublenden oder auszublenden. Dies ist in Verbindung mit der Stylesheet-Angabe zur Anzeige bzw. Nichtanzeige mit Platzhalter – (`visibility`) möglich.

#### Ein Beispiel für Text:

```
<html><head><title>Test</title>
</head><body>
<div id="KommText" style="width:400px; height:100px; font-size:24pt; font-
weight:bold; color:#0000FF; visibility:hidden; filter:blendTrans(Duration=4)">
Hier kommt der Text
</div>
<script language="JScript" type="text/jscript">
<!--
    document.all.KommText.filters.blendTrans.Apply();
    document.all.KommText.style.visibility = "visible";
    document.all.KommText.filters.blendTrans.Play();
//-->
</script>

<div id="GehText" style="width:400px; height:100px; font-size:24pt; font-
weight:bold; color:#0000FF;
visibility:visible; filter:blendTrans(Duration=4)">
Hier geht der Text
</div>
<script language="JScript" type="text/jscript">
<!--
```

```

document.all.GehText.filters.blendTrans.Apply();
document.all.GehText.style.visibility = "hidden";
document.all.GehText.filters.blendTrans.Play();
//-->
</script>
</body></html>

```



21.22: Oben erscheint der Text, unten verschwindet er langsam.

Im obigen Beispiel werden zwei `<div>`-Bereiche mit Text definiert. Beide Bereiche erhalten mit der `id`-Angabe jeweils einen Namen, damit sie per Script ansprechbar sind. Der eine Bereich erhält den Namen `KommText`, der andere den Namen `GehText`. Beide Bereiche erhalten auch mit Hilfe der Stylesheet-Angabe `visibility` eine Angabe dazu, ob sie zunächst angezeigt werden sollen oder nicht. Die Angabe `visibility:hidden` im ersten der beiden `<div>`-Bereiche sorgt dafür, dass dieser Bereich zunächst nicht angezeigt wird. Im zweiten Bereich steht dagegen die Angabe `visibility:visible`. Dadurch wird der entsprechende Text zunächst angezeigt. In beiden `<div>`-Bereichen wird außerdem der `blendTrans`-Filter notiert.

Unterhalb jedes der beiden `<div>`-Bereiche ist ein Script notiert, das den jeweiligen `blendTrans`-Filter ausführt. Das funktioniert genauso wie bei fließenden Übergängen mit dem `blendTrans`-Filter. Der Unterschied ist lediglich, dass dem jeweiligen `<div>`-Bereich zwischen dem Aufruf der Methoden `Apply()` und `Play()` kein anderer Text zugewiesen wird, sondern ein neuer Wert für `visibility`. Der Bereich, der zunächst `visibility:hidden` zugewiesen bekam, wird innerhalb des Scripts mit der Anweisung `document.all.KommText.style.visibility = "visible"` sichtbar. Da der Wechsel zwischen unsichtbar und sichtbar jedoch in den `blendTrans`-Filter eingebunden ist, wird der Text nicht sofort sichtbar, sondern erst durch das Ausführen des `blendTrans`-Filters. Mit dem zweiten Text passiert das Gleiche, nur umgekehrt.

### Ein Beispiel für Grafik

```

<html><head><title>Test</title>
</head><body>
<p><a id="ein" style="display:inline" href="javascript:Einblenden()">Grafik
einblenden</a><br>
<a id="aus" style="display:none" href="javascript:Ausblenden()">Grafik
ausblenden</a></p>

```

```


<script language="JScript" type="text/jscript">
<!--
    function Einblenden()
    {
        document.all.Bild.filters.revealTrans.Transition = 1;
        document.all.Bild.filters.revealTrans.Apply();
        document.all.Bild.style.visibility = "visible";
        document.all.Bild.filters.revealTrans.Play();
        document.all.ein.style.display = "none";
        document.all.aus.style.display = "inline";
    }
    function Ausblenden()
    {
        document.all.Bild.filters.revealTrans.Transition = 0;
        document.all.Bild.filters.revealTrans.Apply();
        document.all.Bild.style.visibility = "hidden";
        document.all.Bild.filters.revealTrans.Play();
        document.all.ein.style.display = "inline";
        document.all.aus.style.display = "none";
    }
//-->
</script>
</body></html>

```

Das voranstehende Beispiel enthält zunächst zwei Verweise. Die beiden Verweise rufen JavaScript-Funktionen auf. Einer der Verweise ruft die Funktion `Einblenden()` zum Einblenden einer Grafik auf, der andere die Funktion `Ausblenden()` zum Ausblenden derselben Grafik. Beide Verweise enthalten auch Stylesheet-Angaben zur Anzeige bzw. Nichtanzeige ohne Platzhalter – (`display`). Der erste Verweis wird zunächst angezeigt, der zweite nicht. Die Angaben werden in den beiden Script-Funktionen `Einblenden()` und `Ausblenden()` dynamisch verändert. So wird ermöglicht, dass der Verweis Grafik einblenden angezeigt wird, wenn die Grafik nicht angezeigt wird, und der Verweis Grafik ausblenden, wenn die Grafik angezeigt wird.

Unterhalb der Verweise wird die Grafik notiert, die ein- und ausblendbar sein soll. Durch die Stylesheet-Angabe `visibility:hidden` wird die Anzeige der Grafik zunächst verhindert. Außerdem erhält die Grafik eine Angabe zum `revealTrans`-Filter. Wenn der Anwender nun auf den zunächst angezeigten Verweis Grafik einblenden klickt, wird die Script-Funktion `Einblenden()` aufgerufen. Diese Funktion ändert zwischen dem Aufruf von `Apply()` und `Play()` des `revealTrans()`-Filters die Anzeige der einzublendenden Grafik auf `visible`. Der Filter wird ausgeführt, und die Grafik wird in der vorgeschriebenen Form (`transition = 1`) eingebildet. Ferner tauscht die Funktion `Einblenden()` noch die Anzeige-Attribute der beiden Verweise aus, sodass nun der Verweis zum Ausblenden der Grafik angezeigt wird. Die Funktion `Ausblenden()` leistet das Gleiche, nur umgekehrt. Dabei wird auch die Art des `revealTrans`-Filters umgekehrt, und zwar durch `transition = 0`.



### 21.6.6 Fließende Seitenübergänge

Als besonderes Bonbon hat sich Microsoft schließlich noch einfallen lassen, die beiden Filter `blendTrans` und `revealTrans` im Zusammenhang mit Meta-Angaben zu interpretieren. Auf diese Weise werden fließende Übergänge zwischen zwei Web-Seiten möglich.



21.23: Von einer Web-Seite zur nächsten blendet der Internet Explorer mit einem Filter weiter.

Beispiel – Datei `uebergang1.htm`:

```
<html><head><title>Test</title>
<meta http-equiv="Page-Exit" content="RevealTrans(Duration=3,Transition=12)">
</head><body bgcolor="#000000" text="#FFFFFF" link="#FFFF66" vlink="#FFFF66"
alink="#FFFFFF">
<h1>Hier ist Seite 1</h1>
<p></p>
<p><a href="uebergang2.htm">und ein Link zu Seite 2</a></p>
</body></html>
```

Beispiel – Datei `uebergang2.htm`:

```
<html><head><title>Test</title>
<meta http-equiv="Page-Enter" content="RevealTrans(Duration=3,Transition=12)">
<meta http-equiv="Page-Exit" content="RevealTrans(Duration=3,Transition=12)">
</head><body bgcolor="#000000" text="#FFFFFF" link="#FFFF66" vlink="#FFFF66"
alink="#FFFFFF">
<h1>Und das hier ist Seite 2</h1>
<p>Sie sieht ein klein wenig anders aus, aber nicht viel.</p>
<p></p>
<p><a href="uebergang1.htm">und das ist ein Link, der wieder zu Seite 1
geht</a></p>
</body></html>
```

Mit `<meta http-equiv="Page-Exit"...` bestimmen Sie einen Filter, der beim Verlassen der Seite angewendet wird, und mit `<meta http-equiv="Page-Enter"...` einen Filter, der beim Laden der Seite angewendet wird. Bei `content=` notieren Sie `RevealTrans` oder `BlendTrans` und dahinter in Klammern die gewünschten Angaben zu `Duration=` und beim `RevealTrans`-Filter `Transition=`. Im obigen Beispiel wird für alle Fälle in beiden Dateien der `RevealTrans`-Filter mit `Transition=12` eingesetzt.

**Beachten Sie:** Im MS Internet Explorer 5.0 war das obige Beispiel nicht nachvollziehbar.

## 22 DHTML – Praxis-Lösungen

### 22.1 Allgemeine DHTML-Bibliothek

Angesichts der vielen verschiedenen DHTML-Modelle, die derzeit noch kursieren, ist es für Programmierer oft nicht einfach, sich die ganzen unterschiedlichen Befehle und Zugriffsmöglichkeiten zu merken. Außerdem werden Dateien bei größeren Scripts ganz schön aufgebläht, wenn man immer wieder die gleichen Browser-Abfragen und Konstrukte für den Elementzugriff notiert. Besser ist es deshalb, einmal eine Bibliothek an allgemein geschriebenen DHTML-Funktionen zu schreiben, diese in eine externe JavaScript-Datei auszulagern, und die JavaScript-Datei dann in denjenigen Dateien mit einzubinden, wo der Code der DHTML-Funktionen benötigt wird.

Das Beispiel in diesem Kapitel stellt eine solche Bibliothek für DHTML-Funktionen vor. Die einzelnen Funktionen werden dabei auch genau erklärt. In den anderen Beispielen in diesem Kapitel kommt die hier erstellte Bibliothek dann zum praktischen Einsatz.

Selbstverständlich können Sie die Funktionenbibliothek für Ihre Zwecke beliebig erweitern. So ist es z.B. denkbar, allgemein geschriebene Funktionen zur CSS-Formatierung oder -Positionierung von Elementen hinzuzufügen. Der hier vorgestellte Grundstock an Funktionen soll vor allem den Zugriff auf Elemente erleichtern und Browser-Abfragen reduzieren.

#### 22.1.1 Quelltext mit Erläuterungen

Der folgende Quelltext zeigt die vollständige DHTML-Bibliothek. Sie finden den kompletten Quelltext auf der Buch-CD in der HTML-Version des Buches. Dann haben Sie die externe JavaScript-Datei mit den DHTML-Funktionen.

Beispiel – Datei *dhtml.js*:

```
/* DHTML-Bibliothek */

var DHTML = 0, DOM = 0, MS = 0, NS = 0, OP = 0;

function DHTML_init() {

  if (window.opera) {
    OP = 1;
  }
  if(document.getElementById) {
    DHTML = 1;
    DOM = 1;
  }
}
```

```

if(document.all && !OP) {
    DHTML = 1;
    MS = 1;
}
if (window.netscape && window.screen && !DOM && !OP){
    DHTML = 1;
    NS = 1;
}
}

function getElem(p1,p2,p3) {
    var Elem;
    if(DOM) {
        if(p1.toLowerCase()=="id") {
            if (typeof document.getElementById(p2) == "object")
                Elem = document.getElementById(p2);
            else Elem = void(0);
            return(Elem);
        }
        else if(p1.toLowerCase()=="name") {
            if (typeof document.getElementsByName(p2) == "object")
                Elem = document.getElementsByName(p2)[p3];
            else Elem = void(0);
            return(Elem);
        }
        else if(p1.toLowerCase()=="tagname") {
            if (typeof document.getElementsByTagName(p2) == "object" ||
                (OP && typeof document.getElementsByTagName(p2) == "function"))
                Elem = document.getElementsByTagName(p2)[p3];
            else Elem = void(0);
            return(Elem);
        }
        else return void(0);
    }
    else if(MS) {
        if(p1.toLowerCase()=="id") {
            if (typeof document.all[p2] == "object")
                Elem = document.all[p2];
            else Elem = void(0);
            return(Elem);
        }
        else if(p1.toLowerCase()=="tagname") {
            if (typeof document.all.tags(p2) == "object")
                Elem = document.all.tags(p2)[p3];
            else Elem = void(0);
            return(Elem);
        }
    }
}

```

```

    else if(p1.toLowerCase()=="name") {
        if (typeof document[p2] == "object")
            Elem = document[p2];
        else Elem = void(0);
        return(Elem);
    }
    else return void(0);
}
else if(NS) {
    if(p1.toLowerCase()=="id" || p1.toLowerCase()=="name") {
        if (typeof document[p2] == "object")
            Elem = document[p2];
        else Elem = void(0);
        return(Elem);
    }
    else if(p1.toLowerCase()=="index") {
        if (typeof document.layers[p2] == "object")
            Elem = document.layers[p2];
        else Elem = void(0);
        return(Elem);
    }
    else return void(0);
}
}

function getCont(p1,p2,p3) {
    var Cont;
    if(DOM && getElem(p1,p2,p3) && getElem(p1,p2,p3).firstChild) {
        if(getElem(p1,p2,p3).firstChild.nodeType == 3)
            Cont = getElem(p1,p2,p3).firstChild.nodeValue;
        else
            Cont = "";
        return(Cont);
    }
    else if(MS && getElem(p1,p2,p3)) {
        Cont = getElem(p1,p2,p3).innerText;
        return(Cont);
    }
    else return void(0);
}

function getAttr(p1,p2,p3,p4) {
    var Attr;
    if((DOM || MS) && getElem(p1,p2,p3)) {
        Attr = getElem(p1,p2,p3).getAttribute(p4);
        return(Attr);
    }
}

```

```

    else if (NS && getElem(p1,p2)) {
        if (typeof getElem(p1,p2)[p3] == "object")
            Attr=getElem(p1,p2)[p3][p4]
        else
            Attr=getElem(p1,p2)[p4]
        return Attr;
    }
    else return void(0);
}

function setCont(p1,p2,p3,p4) {
    if(DOM && getElem(p1,p2,p3) && getElem(p1,p2,p3).firstChild)
        getElem(p1,p2,p3).firstChild.nodeValue = p4;
    else if(MS && getElem(p1,p2,p3))
        getElem(p1,p2,p3).innerText = p4;
    else if(NS && getElem(p1,p2,p3)) {
        getElem(p1,p2,p3).document.open();
        getElem(p1,p2,p3).document.write(p4);
        getElem(p1,p2,p3).document.close();
    }
}

DHTML_init();

```

Die externe JavaScript-Datei darf nichts anderes als JavaScript-Code enthalten, so wie im Beispiel gezeigt. Im Folgenden werden die globalen Variablen und die Funktionen beschrieben. Denn mit diesen Variablen und Funktionen können Sie arbeiten, wenn Sie die Bibliothek verwenden wollen.

### **Globale Variable DHTML**

Diese Variable speichert, ob Dynamisches HTML bei dem vom Anwender eingesetzten Browser verfügbar ist oder nicht. Innerhalb der Funktion `DHTML_init()` wird diese Variable mit einem Wert versorgt, der anschließend nicht mehr geändert werden sollte. Wenn Dynamisches HTML nach irgendeinem der bekannten DHTML-Modelle verfügbar ist, dann erhält die Variable den Wert 1, wenn nicht, dann den Wert 0. Mit der ganz simplen und eleganten Abfrage `if(DHTML)` können Sie dann im Verlauf eines eigenen Scripts überprüfen, ob überhaupt Dynamisches HTML möglich ist.

### **Globale Variable DOM**

Diese Variable speichert, ob Dynamisches HTML nach DOM-Syntax möglich ist oder nicht (Wert 1 oder Wert 0). Innerhalb der Funktion `DHTML_init()` wird diese Variable mit einem entsprechenden Wert versorgt, der anschließend nicht mehr geändert werden sollte. Mit `if(DOM)` können Sie anschließend abfragen, ob ein neuerer Browser am Werk ist, der das DOM bereits in seiner JavaScript-Implementierung unterstützt.

### **Globale Variable MS**

Auch diese Variable wird innerhalb der Funktion `DHTML_init()` mit einem Wert versorgt. Wenn ihr Wert anschließend 1 ist, dann läuft Ihr Script mit ziemlicher Sicherheit auf einem MS Internet Explorer. Handelt es sich um einen Browser der 4er-Generation, ist in der Variablen `DOM` der Wert 0 gespeichert, bei neueren Browsern dagegen 1. Mit `if(MS && !DOM)` können Sie also in Ihrem Script abfragen, ob mit der DHTML-Syntax des Internet Explorers 4.x gearbeitet werden muss.

### **Globale Variable NS**

Wie MS wird auch diese Variable innerhalb der Funktion `DHTML_init()` mit einem Wert versorgt. Wenn ihr Wert anschließend 1 ist, dann läuft Ihr Script mit ziemlicher Sicherheit auf einem Netscape-Browser der 4er-Generation. Denn wenn ein 6er-Netscape oder ein anderer Abkömmling des neu programmierten Mozilla-Browsers eingesetzt wird, ist stattdessen die Variable `DOM` gleich 1, während `NS` gleich 0 ist. Mit `if(NS)` können Sie also in Ihrem Script abfragen, ob mit der DHTML-Syntax des alten 4er-Netscape gearbeitet werden muss.

### **Globale Variable OP**

Wie MS wird auch diese Variable innerhalb der Funktion `DHTML_init()` mit einem Wert versorgt. Wenn ihr Wert anschließend 1 ist, dann läuft Ihr Script mit ziemlicher Sicherheit auf einem Opera-Browser. Mit `if(OP)` können Sie also in Ihrem Script abfragen, ob ein Opera-Browser am Werk ist. Handelt es sich außerdem um einen neueren Opera-Browser, ist in den Variablen `DOM` und in `DHTML` der Wert 1 gespeichert.

### **Funktion `DHTML_init()` – Bibliothek initialisieren**

Aufruf: `DHTML_init();`

Diese Funktion wird automatisch aufgerufen, wenn Sie die hier vorgestellte Bibliothek verwenden wollen. Dazu ist am Ende der Bibliotheksdatei der Aufruf `DHTML_init()` notiert. Anschließend sind die wichtigen globalen Variablen mit ihren Werten versorgt. Wenn `DHTML_init()` nicht vor allen anderen Aufrufen von Funktionen dieser Bibliothek aufgerufen wird, passiert nicht das Gewünschte, da dann eine veraltete Browser-Version angenommen wird. Die Wirkungsweise der übrigen Funktionen ist davon abhängig, dass die globalen Variablen mit korrekten Werten versorgt sind. Durch die Notation am Ende der Datei ist dies jedoch sichergestellt und die globalen Variablen sind in allen Dokumenten verfügbar, die diese Bibliothek einbinden.

Die Funktion `DHTML_init()` bedient sich der Möglichkeit, abzufragen, ob ein Objekt existiert. Bereits in der ersten Abfrage wird getestet, ob ein Opera-Browser vorliegt. Das ist notwendig, da Opera je nach Nutzereinstellung auch angibt, `document.all` oder `document.layers` zu kennen. Zum Testen auf DOM-Verfügbarkeit eignet sich die Frage, ob `document.getElementById` verfügbar ist, denn das ist eine wichtige DOM-Methode, die in allen DOM-Implementierungen existieren sollte.

Zum Testen auf die ältere Microsoft- und Netscape-Syntax wird auf deren zentrale DHTML-Objekte hin abgefragt. Wenn `document.all` verfügbar ist, ist Syntax nach dem älteren Microsoft-Modell angesagt. Wenn `document.layers` verfügbar ist, Syntax nach dem älteren Net-

scape-Modell. In beiden Fällen wird mit !OP der Opera-Browser ausgeschlossen für den Fall, dass die Nutzereinstellungen die Kenntnis dieser Objekte ermöglichen.

### **Funktion `getElem()` – Elementobjekt ermitteln**

Beispielaufrufe:

```
Elementobjekt = getElem("id","Hinweis",null);
Elementobjekt = getElem("name","Eingabefeld",0);
Elementobjekt = getElem("tagname","p",8);
Layerobjekt = getElem("index",2,null,null);
```

Diese Funktion ist eigentlich die zentrale Funktion der hier vorgestellten Bibliothek. Sie greift auf ein bestimmtes Element im Dokument zu und gibt das Objekt dieses Elements zurück. Das funktioniert mit DOM-fähigen Browsern ebenso wie mit dem älteren Internet Explorer 4.x und in eingeschränkter Form (nur im Zusammenhang mit `layer`- oder `div`-Elementen) auch mit Netscape 4.x.

In den meisten Fällen müssen Sie diese Funktion nicht selber aufrufen. Es handelt sich vielmehr um eine Basisfunktion, die von den übrigen Funktionen der Bibliothek benutzt wird. Dennoch gibt es auch Fälle, in denen ein Direktaufruf dieser Funktion Sinn macht. Die Funktion erwartet folgende Parameter:

- `p1` ist die Art und Weise, wie auf ein Element zugegriffen werden soll. Es gibt vier erlaubte Werte für diesen Parameter: `"id"`, `"name"`, `"tagname"` und `"index"`. Übergeben Sie `"id"`, wenn Sie auf ein Element zugreifen wollen, das ein `id`-Attribut hat, z.B. `<div id="Bereich">`. Übergeben Sie `"name"`, wenn das Element, auf das Sie zugreifen wollen, kein `id`-Attribut, aber dafür ein `name`-Attribut hat, z.B. `<a name="Anker">`. Übergeben Sie `"tagname"`, wenn das Element, auf das Sie zugreifen wollen, weder ein `id`- noch ein `name`-Attribut hat. Übergeben Sie `"index"` nur dann, wenn es sich um Netscape-Layer handelt, die über das `layers`-Objekt mit Hilfe von Indexnummern angesprochen werden sollen.
- `p2` ist die nähere Angabe zu dem, was Sie bei `p1` festgelegt haben: Wenn Sie bei `p1` den Wert `"id"` übergeben, dann erwartet die Funktion bei `p2` den Wert des `id`-Attributs. Wenn Sie bei `p1` den Wert `"name"` übergeben, dann erwartet `p2` den Wert des `name`-Attributs. Wenn Sie bei `p1` den Wert `tagname` übergeben haben, dann erwartet `p2` den gewünschten Elementnamen, also z.B. `h1` oder `div`. Wenn Sie bei `p1` den Wert `"index"` übergeben haben, erwartet `p2` die Indexnummer für den gewünschten Netscape-Layer.
- `p3` wird nur dann benötigt, wenn bei `p1` einer der beiden Werte `"name"` oder `"tagname"` angegeben wurde. In diesem Fall müssen Sie bei `p3` eine Indexnummer angeben, mit der Sie festlegen, das wievielte Element im Dokument mit dem entsprechenden Namen oder Elementnamen Sie meinen. Das erste Element hat die Indexnummer 0, das zweite Nummer 1 usw.

Übergeben Sie bei Parametern, für die Sie keinen Wert angeben, einfach das Wort `null` (ohne Anführungszeichen!).

Die Funktion `getElem()` geht der Reihe nach die möglichen DHTML-Modelle durch und fragt zuerst mit `if(DOM)` die globale Variable `DOM` daraufhin ab, ob DOM-Syntax verfügbar ist. Wenn ja, dann wird abhängig von den übergebenen Parametern das gewünschte Element nach DOM-



Syntax ermittelt und zurückgegeben. Nur wenn keine DOM-Syntax verfügbar ist, wird versucht, die älteren DHTML-Modelle anzuwenden. Zunächst wird mit `if(MS)` geprüft, ob das ältere DHTML-Modell von Microsoft verfügbar ist. Wenn ja, wird das Element nach dessen Syntax über das `document.all`-Objekt ermittelt. Wenn auch die Microsoft-Syntax nicht greift, bleibt noch die ältere Netscape-Syntax, die mit `if(NS)` abgefragt wird. In diesem Fall wird aus den übergebenen Parametern der Zugriff auf ein Layer-Objekt konstruiert.

Gleichzeitig erfolgt in der Funktion eine Fehlerbehandlung. In jedem Zweig der Funktion wird mit dem `typeof`-Operator geprüft, ob das gewünschte Element überhaupt existiert und ob dieses ein Objekt darstellt. Tritt dieser Fall ein, wird das Element erzeugt und zurückgegeben. Andernfalls wird mit Hilfe des `void`-Operators *undefined* zurückgegeben.

### ***Funktion getCont() – Elementinhalt ermitteln***

Beispielaufufe:

```
if(! NS) Elementinhalt = getCont("id","Hinweis",null);
if(! NS) Elementinhalt = getCont("name","Verweis",3);
if(! NS) Elementinhalt = getCont("tagname","p",8);
```

Über diese Funktion, die ihrerseits auf die Funktion `getElem()` zurückgreift, können Sie auf bequeme Weise den Inhalt eines Elements ermitteln – zumindest wenn das DOM-Modell oder das ältere Microsoft-Modell für DHTML verfügbar sind. Das ältere Netscape-Modell kennt kein Auslesen von Elementinhalten. In den Beispielaufrufen ist der Funktionsaufruf deshalb davon abhängig gemacht, dass nicht nach älterer Netscape-Syntax gearbeitet wird (`if(!NS)`).

Die Funktion liefert bei Verwendung des DOM-Modells eine leere Zeichenkette zurück, falls der Inhalt des Elements kein Text ist, sondern ein weiteres Element. Wenn also beispielsweise notiert ist:

```
<p>der Text</p>
```

dann liefert die Funktion den Wert der `Text` zurück. Wenn aber notiert ist:

```
<p><b>der Text</b></p>
```

dann liefert die Funktion bei Anwendung des DOM-Modells eine leere Zeichenkette zurück, da es keinen Text als Elementinhalt von `p` gibt, sondern der Inhalt ein anderes Element, nämlich ein `b`-Element ist.

Die Parameter sind bei `getCont()` die gleichen wie bei `getElem()`. Nur der Wert "index" bei Parameter `p1` wird von dieser Funktion nicht verarbeitet. Übergeben Sie bei Parametern, für die Sie keinen Wert angeben, das Wort `null`.

Die Funktion `getCont()` bedient sich also der Basisfunktion `getElem()` und reicht ihre drei übergebenen Parameter einfach an jene durch. Bei DOM-Syntax versucht sie, an den Textinhalt des Elements mit `firstChild.nodeValue` zu kommen, bei Microsoft-Syntax über die Eigenschaft `innerText` des `all`-Objekts. An der Art, wie die Funktion `getCont()` die Funktion `getElem()` aufruft, können Sie auch sehen, wie Sie die Basisfunktion `getElem()` anderweitig direkt aufrufen können. Da diese Funktion ein Elementobjekt zurückgibt, können Sie direkt Eigenschaften und Methoden eines solchen Objekts an den Funktionsaufruf hängen. Allerdings müssen Sie dazu selber abfragen, nach welchem Modell gearbeitet wird. Wenn nach DOM-

Modell gearbeitet wird (abfragbar mit `if(DOM)`), dürfen beim Aufruf von `getElem()` Methoden und Eigenschaften der HTML-Elementobjekte angehängt werden oder auch solche des `node-`Objekts.

Die Funktion `getCont()` kümmert sich auch um Browser wie Opera, die Inhalte noch nicht dynamisch lesen können, jedoch andere Teile der Bibliothek interpretieren. Bevor der Versuch unternommen wird, Inhalte zu lesen, fragt sie mit `if (DOM && getElem(p1,p2,p3) && getElem(p1,p2,p3).firstChild)` ab, ob das gewünschte Objekt existiert und ob auch das Objekt `firstChild` vorhanden ist. Erst wenn diese Bedingung erfüllt ist, wird versucht, den Inhalt zu lesen. Dadurch wird in jedem Fall eine Fehlermeldung vermieden.

### ***Funktion `getAttr()` – Attributinhalt ermitteln***

Beispielaufrufe:

```
Attribut=getAttr("name","Bildname",0,"height");
Attribut=getAttr("id","Bereich",null,"id");
if (!NS) Attribut=getAttr("name","Anker",0,"href")
if (NS) Attribut=getAttr("name","links",0,"href")
```

Über diese Funktion, die ihrerseits auf die Funktion `getElem()` zurückgreift, können Sie auf bequeme Weise den Wert eines Attributs in einem Element ermitteln – zumindest wenn das DOM-Modell oder das ältere Microsoft-Modell für DHTML verfügbar sind. Für einige Objekte ist es auch im Netscape 4 möglich. Neben den drei bereits bekannten ersten Parametern erwartet diese Funktion einen vierten Parameter `p4`. Bei diesem müssen Sie den Namen des gewünschten Attributs übergeben. Übergeben Sie bei Parametern, für die Sie keinen Wert angeben, das Wort `null`.

Die Funktion `getAttr()` benutzt die Objektmethoden `getAttribute()`, die sowohl im DOM-Modell als auch im älteren Microsoft-Modell verfügbar sind.

Bei Netscape 4 dagegen wird die gewünschte Eigenschaft direkt abgefragt. Bei diesem Browser ist es in einigen Fällen nur möglich, über Unterobjekte auf die gewünschte Eigenschaft zuzugreifen, z.B. bei Verweisen und Ankern. Sie können darauf zugreifen, indem Sie im Parameter `p2` den Namen des Unterobjektes (z.B. `links` oder `elements`) und im Parameter `p3` den Namen bzw. die Indexnummer des Objektes notieren.

Die Auswahl des richtigen Objekts übernimmt die Basisfunktion `getElem()`. Natürlich erfolgt auch in jedem Fall die Überprüfung, ob das gewünschte Objekt überhaupt existiert.

### ***Funktion `setCont()` – Elementinhalt neu dynamisch ändern:***

Beispielaufrufe:

```
setCont("id","Hinweis",null,"neuer Hinweistext");
setCont("name","Verweis",3,"neuer Verweistext");
setCont("tagname","p",8,"neuer Textinhalt des Elements");
if(NS) setCont("index","2",null,"neuer Inhalt des Layers");
```

Mit dieser Funktion können Sie den Inhalt eines Elements dynamisch ändern. Die Funktion `setCont()` bedient sich für den Zugriff auf das gewünschte Element der Basisfunktion

`getElem()`. Die ersten drei Parameter, die bei dieser Funktion die gleichen sind wie bei den übrigen Funktionen, werden dazu an `getElem()` weitergereicht. Als vierten Parameter `p4` erwartet die Funktion `setCont()` den gewünschten Textinhalt. Beachten Sie, dass dieser Text zumindest beim DOM-Modell und beim älteren Microsoft-Modell nur als reiner Text interpretiert wird. HTML-Formatierungen im Text werden dabei als Klartext interpretiert.

Übergeben Sie bei Parametern, für die Sie keinen Wert angeben, das Wort `null`. Bei Netscape-Layern benutzt die Funktion `setCont()` automatisch die üblichen Befehle zum Öffnen des Layer-Dokuments (`document.Layer.document.open()`), zum Schreiben (`document.Layer.document.write()`) und zum Schließen (`document.Layer.document.close()`).

Die Funktion `setCont()` kümmert sich auch um Browser wie Opera, die Inhalte noch nicht dynamisch ändern können, jedoch andere Teile der Bibliothek interpretieren. Bevor der Versuch unternommen wird, Inhalte zu ändern, fragt sie mit `if (DOM && getElem(p1,p2,p3) && getElem(p1,p2,p3).firstChild)` ab, ob das gewünschte Objekt existiert und ob auch das Objekt `firstChild` existiert. Erst wenn diese Bedingung erfüllt ist, versucht sie den Inhalt zu ändern. Dadurch wird in jedem Fall eine Fehlermeldung vermieden.

### 22.1.2 Test der DHTML-Bibliothek

Das folgende Beispiel zeigt, wie Sie die DHTML-Bibliothek verwenden können. Die einzelnen Funktionen werden in dieser Beispieldatei mit konkreten Beispielen aufgerufen. Am besten können Sie die Funktionsweise testen, wenn Sie das Anzeigebeispiel mal mit verschiedenen Browsern aufrufen. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript" src="dhtml.js"></script>
<script type="text/javascript">
<!--
function Testen() {
    var Ausgabertext = "";
    if(DHTML) Ausgabertext += "DHTML ist aktiviert!\n"
    else      Ausgabertext += "DHTML ist nicht aktiviert!\n";
    if(DOM)   Ausgabertext += "DOM wird unterstützt!\n";
    else      Ausgabertext += "DOM wird nicht unterstützt!\n";
    if(MS)    Ausgabertext += "Alte Microsoft-Technologie ohne DOM!\n";
    if(NS)    Ausgabertext += "Alte Netscape-Technologie ohne DOM!\n";

    h1_Inhalt = getCont("tagname","h1",0);
    if(h1_Inhalt) Ausgabertext += "Inhalt des h1-Elements: " + h1_Inhalt + "!\n";
    else          Ausgabertext += "Inhalt des h1-Elements konnte nicht ermittelt
werden!\n";

    p_Inhalt = getCont("id","Hinweisabsatz",null);
    if(p_Inhalt) Ausgabertext += "Inhalt des p-Elements: " + p_Inhalt + "!\n";
    else          Ausgabertext += "Inhalt des p-Elements konnte nicht ermittelt
werden!\n";
```

```

    if (NS) Verweis = getAttr("name","anchors","Anker","text");
    else Verweis = getCont("name","Anker",0);
    if(Verweis)    Ausgabertext += "Verweistext: " + Verweis + "!\n";
    else          Ausgabertext += "Verweistext konnte nicht ermittelt werden!\n";

    h1_align = getAttr("tagname","h1",0,"align");
    if(h1_align)  Ausgabertext += "Ausrichtung des h1-Elements: " + h1_align +
"!\\n";
    else          Ausgabertext += "Ausrichtung des h1-Elements konnte nicht
ermittelt werden!\\n";

    Bild_hoehe= getAttr("name","auge",0,"height");
    if(Bild_hoehe) Ausgabertext += "Bildhöhe: " + Bild_hoehe + "px.\\n";
    else          Ausgabertext += "Die Bildhöhe konnte nicht ermittelt werden!\\n";

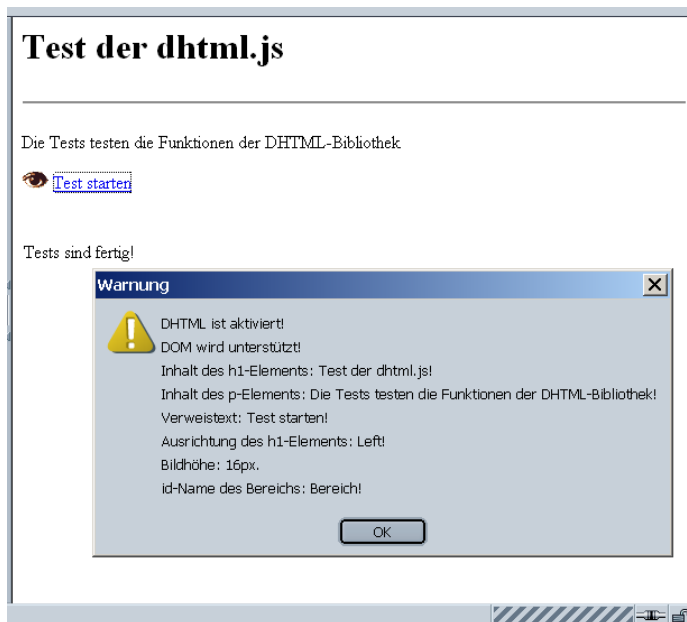
    Bereich_id = getAttr("id","Bereich",null,"id");
    if(Bereich_id) Ausgabertext += "id-Name des Bereichs: " + Bereich_id + "!\n";
    else          Ausgabertext += "id-Name des Bereichs konnte nicht ermittelt
werden!\\n";

    setCont("id","Bereich",null,"Tests sind fertig!");

    alert(Ausgabertext);
}
//-->
</script>
<style type="text/css">
<!--
#Bereich {position:absolute; top:200px; left:10px; width:500px; height:100px; }
-->
</style>
</head><body>
<h1 align="left">Test der dhtml.js</h1>
<hr noshade size="1">
<p id="Hinweisabsatz">Die Tests testen die Funktionen der DHTML-Bibliothek</p>
<p>
<a name="Anker" href="javascript:Testen()">Test starten</a></p>
<div id="Bereich">&nbsp;</div>
</body></html>

```

Das Beispiel enthält im sichtbaren Bereich der HTML-Datei eine Überschrift erster Ordnung, einen Textabsatz, einen weiteren Textabsatz mit einem Verweis, vor dem außerdem eine Grafik eingebunden ist, und schließlich einen `div`-Bereich, der mit Hilfe von CSS absolut positioniert ist. Die entsprechenden CSS-Angaben zur Positionierung stehen im Dateikopf in einem `style`-Bereich, wo die Formate für `#Bereich` notiert sind. Generell ist es wegen einiger Schwächen von Netscape 4.x besser, die CSS-Formatierungen der simulierten Layer in einem zentralen Stylesheet gleich im Dateikopf zu notieren und nicht im einleitenden `<div>`-Tag.



22.1: Das Ergebnis der Testroutine bei Netscape 6.2.

Im Dateikopf sind ferner zwei JavaScript-Bereiche definiert. Der erste ist leer und bindet lediglich die DHTML-Bibliothek ein, die in der externen JavaScript-Datei *dhtml.js* gespeichert ist. Der zweite Script-Bereich enthält den JavaScript-Code, der die Testaufrufe durchführt.

Das JavaScript innerhalb des Dateikopfes wird aufgerufen, wenn der Anwender auf den Verweis klickt, der in der Datei notiert ist. Dann wird die Funktion `Testen()` aufgerufen. Diese Funktion wie auch das gesamte JavaScript kann ohne weitere Umwege auf alle globalen Variablen und die Funktionen der DHTML-Bibliothek zugreifen. Zunächst greift die Funktion `Testen()` auf die initialisierten globalen Variablen der DHTML-Bibliothek zu und prüft, welches DHTML-Modell verfügbar ist. Abhängig davon, was verfügbar ist, wird die Variable `Ausgabertext` um einen Text erweitert.

Als Nächstes wird mit der Funktion `getCont()` aus der DHTML-Bibliothek gespielt. Mit `getCont("tagname", "h1", 0)` wird via Elementname auf das erste h1-Element (Indexzähler 0) zugegriffen. Der Rückgabewert von `getCont()` wird in der Variablen `h1_Inhalt` gespeichert. Abhängig davon, ob diese nach dem Funktionsaufruf eine Zeichenkette mit einer Länge größer 0 enthält (`if(h1_Inhalt)`), wird die Variable `Ausgabertext` wieder um einen weiteren Text erweitert.

Zwei weitere Aufrufe von `getCont()` folgen, die andere Varianten der Parameterversorgung ausprobieren. Mit `getCont("id", "Hinweisabsatz", null)` wird auf das Element mit dem Attribut `id="Hinweisabsatz"` zugegriffen. Mit `getCont("name", "Anker", 0)` wird auf das erste Element im Dokument zugegriffen, bei dem das Attribut `name="Anker"` notiert ist. Da auch mit Netscape 4 auf den Text eines Verweises zugegriffen werden kann, wird für diesen Browser nicht die Funktion `getCont()` aufgerufen, sondern `getAttr()`. Berücksichtigt wurde

zudem, dass sobald ein Verweis gleichzeitig ein Anker ist, der Verweistext nur über das Anchor-Objekt ausgelesen werden kann.

Daran anschließend probiert die Funktion `Testen()` die Bibliotheksfunktion `getAttr()` mit drei Aufrufen aus, einmal über den Elementnamen, einmal über den Tag-Namen und einmal über das `id`-Attribut. Schließlich wird auch noch `setCont()` aufgerufen. Damit es in allen Browsern inklusive Netscape 4.x funktioniert, wird der Inhalt des absolut positionierten `div`-Bereichs neu gesetzt, den Netscape als Layer erkennt. Am Ende gibt die Funktion `Testen()` die gesammelten Einzelergebnisse, die sie in der Variablen `Ausgabertext` zusammengetragen hat, in einem Meldungsfenster aus.

### 22.1.3 DHTML-Bibliothek erweitern

Die hier vorgestellte DHTML-Bibliothek leistet nur einige wenige Grundfunktionen, erleichtert den Zugriff auf Elemente etwas und macht in vielen Fällen den JavaScriptCode etwas übersichtlicher und besser lesbar. Es gibt aber noch vieles, was sie nicht leistet. Wenn Sie häufig mit Dynamischem HTML arbeiten, empfiehlt es sich, die DHTML-Bibliothek um eigene, andere Funktionen zu erweitern. Wichtig ist bei so einer Funktionen-Bibliothek, dass alle Funktionen allgemein geschrieben sind. Deshalb erwarten die Funktionen in der Regel mehrere Parameter, um ihren Input für die zu verarbeitenden Daten zu erhalten.

Denkbar wären beispielsweise noch Funktionen, die direkt auf CSS-Eigenschaften von Elementen zugreifen, oder Funktionen, die in der Lage sind, ein Element dynamisch neu zu positionieren. Ausgehend davon sind auch Funktionen möglich, die bestimmte Bewegungsabläufe wie Kreisbahnen usw. berechnen und Elemente entsprechend positionieren. Andere denkbare Funktionen könnten sich dem leider noch ziemlich komplizierten, weil sehr unterschiedlich implementierten Event-Handling widmen und dort einiges vereinfachen.

Eine Funktionen-Bibliothek sollte andererseits jedoch nicht zu groß werden. Denn wenn Sie am Ende zig Kilobyte externes JavaScript einbinden, um drei triviale Anweisungen zu notieren, dann ist eine Funktionen-Bibliothek nicht angemessen. In solchen Fällen ist es besser, die paar Befehle ohne Zuhilfenahme einer Funktionen-Bibliothek zu notieren. Achten Sie stets darauf, dass Ihre Bibliotheken eine Fehlerhandlung durchführen und jeweils die Verfügbarkeit der angesprochenen Objekte überprüft.

## 22.2 Dynamische Navigationsleiste

### 22.2.1 Hinweise zu diesem Beispiel

Auf vielen Web-Seiten nimmt die Navigation eigentlich viel zu viel Platz weg. Da wird oft ein Drittel des Anzeigefensters geopfert, um eine oder zwei Handvoll Links zu anderen Projektseiten anzubieten. Mit diesen Platzproblemen macht das Beispiel auf dieser Seite Schluss. Bei dieser Lösung gehört die Web-Seite der freien inhaltlichen Gestaltung und dem Text. Die Navigation blendet sich nur dann ein, wenn der Anwender es will. Dazu muss er auch nicht extra irgendwo hinklicken. Er bewegt einfach die Maus Richtung linken Fensterrand, und schon blendet sich dort die Navigationsleiste ein. Bewegt er die Maus wieder weg vom linken Fensterrand, blendet

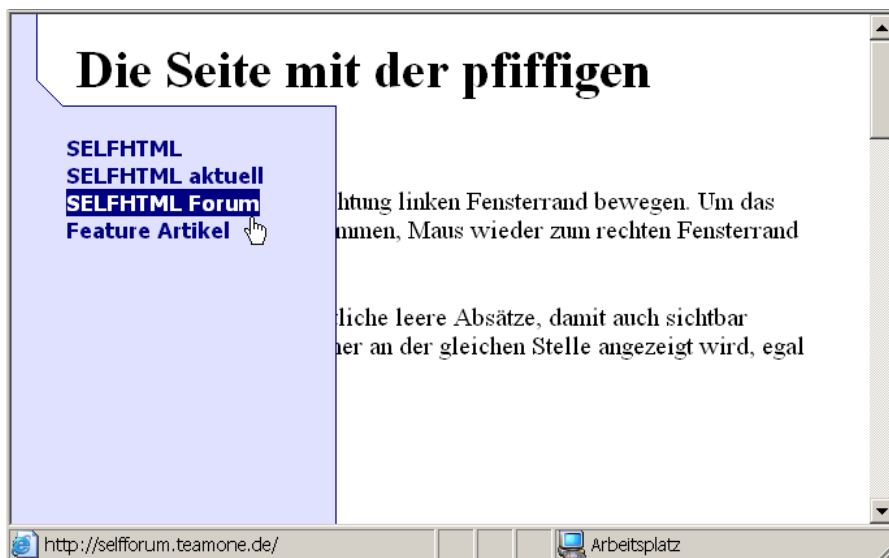
sich die Navigationsleiste automatisch wieder aus. Diesen Effekt kennen viele Anwender in ähnlicher Form von der Windows-Taskleiste und sind deshalb damit vertraut.

Das Beispiel funktioniert mit den Browsern ab der 4er-Generation. Das Beispiel zeigt vor allem, wie man das Problem der unterschiedlichen Event-Handling-Modelle bei den Browsern in den Griff bekommen kann. Das Beispiel benutzt ferner die DHTML-Bibliothek, um auf Elemente zuzugreifen.

### 22.2.2 Quelltext mit Erläuterungen

Das Beispiel zeigt eine vollständige HTML-Datei mit der dynamischen Navigation. Wenn Sie die Navigation auf mehreren Seiten haben wollen, müssen Sie den JavaScript-Code entweder in alle Seiten einbauen, oder Sie lagern ihn in eine externe JavaScript-Datei aus und binden diese einfach nur ein.

Die Datei *dhtml.js*, in der die Funktionen der DHTML-Bibliothek enthalten sind, muss vorhanden sein und sich im Beispiel im gleichen Verzeichnis befinden wie die HTML-Datei.



22.2: Die Navigationsleiste klappt erst auf, wenn der Mauszeiger in den linken Fensterbereich kommt.

Das Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript" src="dhtml.js"></script>
<script type="text/javascript">
<!--
function Menue() {
  if(DOM) {
    if(MS)
```

```

    getElem("id","Nav",null).style.top = document.body.scrollTop + 50;
else
    getElem("id","Nav",null).style.top = window.pageYOffset + 50;
}
if(DOM || MS) {
    if (!DOM) getElem("id","Nav",null).style.top = document.body.scrollTop + 50;
    if (OP) getElem("id","NavLinks",null).style.pixelTop = NavLinksPos;
    getElem("id","Nav",null).style.visibility = "visible";
}
else if(NS) {
    getElem("id","Nav",null).visibility = "show";
    getElem("id","Nav",null).top = window.pageYOffset + 50;
}
}

function noMenue() {
    if(DOM || MS)
        getElem("id","Nav",null).style.visibility = "hidden";
    if(NS)
        getElem("id","Nav",null).visibility = "hide";
}

function handleMove(ev) {
    if(!MS) {
        Event = ev;
        if(Event.screenX < 270)
            Menue();
        else if(Event.screenX > 420)
            noMenue();
    }
}

function MShandleMove() {
    if(MS) {
        if(window.event.clientX < 270)
            Menue();
        else if(window.event.clientX > 420)
            noMenue();
    }
}

function Event_init() {
    if(DOM && !MS && !OP) {
        getElem("tagname","body",0).addEventListener("mousemove", handleMove, true);
    }
    if(NS) {
        document.captureEvents(Event.MOUSEMOVE);
    }
}

```



```

    document.onmousemove=handleMove;
}
if (DOM && OP) {
    document.onmousemove=handleMove;
    NavLinksPos=42; //Position des Bereiches NavLinks
    getElem("id","NavLinks",null).style.pixelTop=NavLinksPos;
}
if (MS) getElem("tagname","body",0).onmousemove=MShandleMove;
}
//-->
</script>
<style type="text/css">
<!--
body { margin-left:50px; margin-top:20px; margin-right:50px; }
#Nav { position:absolute; top:50px; left:0px; padding:0px; visibility:hidden;
margin:0px; }
#NavLinks { position:absolute; top:42px; left:42px; padding:0px; }
a.nav:link { color:#000080;
    font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
a.nav:visited { color:#000080;
    font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
a.nav:hover { color:#FFFFFF; background-color:#000080;
    font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
a.nav:active { color:#000080;
    font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
-->
</style>
</head>
<body onLoad="Event_init()" background="navigation_back.gif" bgcolor="#FFFFFF">

<div id="Nav">

<div id="NavLinks">
<style type="text/css">
<!--
    a.nav:link { color:#000080;
        font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
    a.nav:visited { color:#000080;
        font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
    a.nav:hover { color:#FFFFFF; background-color:#000080;

```

```

        font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
        a.nav:active { color:#000080;
            font-family:Tahoma,sans-serif; font-weight:bold; font-size:10pt; text-
decoration:none; }
        -->
    </style>
    <a class="nav" href="http://selfhtml.teamone.de/"><b>SELFHTML</b></a><br>
    <a class="nav" href="http://selfaktuell.teamone.de/"><b>SELFHTML
aktuell</b></a><br>
    <a class="nav" href="http://selfforum.teamone.de/"><b>SELFHTML Forum</b></a><br>
    <a class="nav" href="http://selfaktuell.teamone.de/artikel/"><b>Feature
Artikel</b></a><br>
</div>
</div>

<h1>Die Seite mit der pfiffigen Navigation</h1>

<p>Einfach mit die Maus Richtung linken Fensterrand bewegen.
Um das Menü wieder wegzubekommen, Maus wieder zum rechten Fensterrand
bewegen.</p>

<p>Und jetzt kommen noch etliche leere Absätze, damit auch sichtbar wird,
dass das Menü immer an der gleichen Stelle angezeigt wird, egal wie weit
man
scrollt.</p>

<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p><p>&nbsp;</p>
<p>Na, das dürfte reichen :-)</p>

</body></html>

```

Im sichtbaren HTML-Bereich der Datei ist zunächst ein größerer `div`-Bereich notiert, der mit Hilfe von CSS-Eigenschaften absolut positioniert ist (die Angaben dazu stehen im `style`-Bereich im Dateikopf) und dabei auch die Angabe `visibility:hidden` enthält. Das bedeutet, alles, was sich in diesem Bereich befindet, wird zunächst einmal gar nicht angezeigt. Innerhalb dieses Bereichs mit dem `id`-Namen `Nav` ist zunächst eine Grafik notiert (*navigation.gif*) und anschließend ein weiterer, absolut positionierter `div`-Bereich, wobei aber dort CSS-Zuweisungen wie `top:42px` oder `left:42px` als relativ zur oberen linken Ecke des Eltern-`div`-Bereichs zu inter-

pretieren sind. Die Grafik *navigation.gif* ist einfach eine optische Unterlage für den nachfolgenden `div`-Bereich, der die eigentlichen Verweise der Navigationsleiste enthält.

Der Rest des in der Datei notierten statischen HTML-Codes ist für das Beispiel irrelevant. Die vielen Textabsätze mit dem erzwungenen Leerzeichen `&nbsp;`; dazwischen dienen nur dazu, viel Dokumenthöhe zu erzeugen, sodass man in der Datei scrollen kann. Denn es soll demonstriert werden, dass die dynamische Navigationsleiste immer an der gleichen Stelle erscheint, auch wenn im Dokument gescrollt wird.

Im einleitenden `<body>`-Tag der Datei ist der Event-Handler `onLoad` notiert. Wenn das Dokument vollständig geladen ist, wird die Funktion `Event_init()` aufgerufen. `Event_init()` stößt die Überwachung der Mausbewegungen im Dokument an. Denn nur wenn eine solche Überwachung stattfindet, kann die Navigationsleiste dynamisch eingeblendet werden.

Event-Handling ist leider ein schwieriges Kaptiel, da die Implementierungen bislang doch noch recht unterschiedlich sind. Das Beispiel versucht es erst mal nach der neueren DOM-Syntax. Die Abfrage `if(DOM && !MS && !OP)` stellt sicher, dass der Internet Explorer und Opera 5.12 in diesen Ausführungszweig noch nicht gelangen, da beide in der Version 5.x zwar DOM-Elementzugriffe, aber noch kein Event-Handling nach DOM-Syntax interpretieren. In den entsprechenden Zweig gelangt aber Netscape 6.x, der die Methode `addEventListener()` versteht, mit der nach DOM-Syntax ein Funktion bestimmt wird, die aufgerufen wird, wenn ein bestimmtes Ereignis eintritt. Im Beispiel wird das Ereignis `mousemove` überwacht. Während der Anwender im Dokumentbereich die Maus bewegt, bewirkt diese »Registrierungs-Anweisung«, dass dann so oft es geht die Funktion `handleMove()` aufgerufen wird. Zur Bestimmung des zu überwachenden Elementes wird die Funktion `getElement()` der DHTML-Bibliothek aufgerufen.

Mit `if(NS)` fragt die Funktion `Event_init()` ab, ob ein Netscape-Browser der 4er-Generation am Werk ist. Dort gibt es ebenfalls eine »Registrierung« für Ereignisse. Mit `document.captureEvents(Event.MOUSEMOVE)` wird das Ereignis »Mausbewegung« registriert, und mit `document.onmousemove=handleMove` wird die Funktion `handleMove()` aufgerufen, wenn das Ereignis eintritt.

Der Anweisung `if (DOM && OP)` folgen DHTML-fähige Opera-Browser. Hier erfolgt die Initialisierung der Event-Überwachung mit `document.onmousemove=handleMove`. Da Opera 5 unter Windows offensichtlich die Position des `div`-Bereich `NavLinks` vergisst, wird die im Stylesheet-Bereich festgelegte Position in der Variablen `NavLinksPos` gespeichert. Anschließend wird der Bereich `NavLinks` neu positioniert. Damit auch ältere Versionen von Opera 5 damit zurecht kommen, wird die Eigenschaft `pixelTop` verwendet. Zur Bestimmung des Elementes wird die Funktion `getElement()` der DHTML-Bibliothek aufgerufen.

Beim Internet Explorer gestaltet sich die Ereignisbehandlung leider etwas anders. Mit `if (MS)` wird in der Funktion `Event_init()` der für den MS Internet Explorer gültige Bereich abgearbeitet. Auch hier wird wieder die Funktion `getElement()` der DHTML-Bibliothek verwendet. Im Gegensatz zu den anderen Browsern wird hier die Funktion `MShandleMove()` aufgerufen, wenn das Ereignis eintritt. Dort wird mit `if(MS)` sichergestellt, dass kein anderer Browser als ein Internet Explorer am Werk ist. Abhängig davon wird mit `window.event.clientX` der Abstand des Mauszeigers vom linken Fensterrand ermittelt. Wenn er geringer ist als 270 Pixel, wird die Funktion `Menue()` aufgerufen. Wenn er höher als 420 Pixel ist, die Funktion `noMenue()`. Genau das Gleiche passiert auch in der Funktion `handleMove()`, in die die Netscape-Browser gelangen. Dort wird der Abstand des Mauszeigers vom linken Fensterrand allerdings mit der `screenX`-

Eigenschaft abgefragt. Das sind die tausend kleinen Unterschiede zwischen den Browsern, die Programmierer zum Wahnsinn treiben können. Immerhin wird irgendwann diese ganze `if`-Weiche entfallen und alle Browser werden die DOM-Syntax verstehen.

Die Funktion `Menue()` blendet die Navigationsleiste ein, und die Funktion `noMenue()` sorgt dafür, dass die Navigationsleiste nicht oder nicht mehr sichtbar ist. Beide Funktionen führen zu diesem Zweck wieder diverse `if`-Abfragen zur DHTML-Modellunterscheidung durch und ändern letztendlich hauptsächlich die CSS-Eigenschaft `visibility` (*Sichtbarkeit*) des großen `div`-Bereichs mit dem Id-Namen `Nav`. Die Funktion `Menue()` sorgt allerdings außerdem dafür, dass der von ihr auf sichtbar geschaltete Navigationsbereich auch immer schön im sichtbaren Bereich des Dokuments angezeigt wird, egal, wohin der Anwender im Dokument gescrollt hat. Dazu muss die Funktion die aktuelle Pixelposition der »Klopapierrolle« ermitteln, die ein langes Dokument darstellt. Auch dabei ist die Syntax leider wieder völlig unterschiedlich bei den Browsern. Beim Internet Explorer kommt die Funktion über die Eigenschaft `document.body.scrollTop` an den gewünschten Wert, und bei Netscape-Browsern und Opera über `window.pageYOffset`. Zu dem ermittelten Wert werden noch mal 50 Pixel dazugezählt, denn in dem `div`-Bereich mit dem Id-Namen `Nav` ist ja als `top`-Angabe, also als Startposition von oben gesehen, ebenfalls der Wert 50 angegeben.

Da Opera 5 beim Ändern der Fenstergröße mit JavaScript gesetzte Positionen »vergisst«, wird vor jedem Einblenden des Menüs dem Bereich `NavLinks` die in der Variablen `NavLinksTop` gespeicherte Position zugewiesen.

Vielleicht fragen Sie sich jetzt noch, woher die Werte 270 und 420 für die Mausposition vom linken Fensterrand kommen, die über den Aufruf von `Menue()` und `noMenue()` entscheiden. Dazu kann man nicht viel mehr sagen, als dass es mit verschiedenen Browsern ausgetestete Erfahrungswerte sind. In der Regel funktioniert es beispielsweise nicht mehr, wenn der kleinere Wert kleiner als 250 ist. Warum, bleibt rätselhaft. Aber diese Rätselhaftigkeiten gehören leider zum Alltag der DHTML-Programmierung, solange die JavaScript-Interpreter der Browser immer noch nicht ausgereift sind.

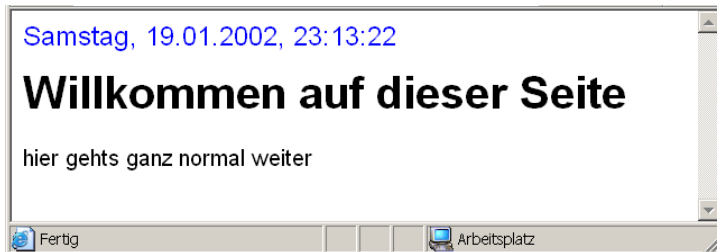
**Beachten Sie:** Erklären Sie dem Anwender auf der ersten Seite, wo Sie eine solche Navigation verwenden, kurz mit einem Satz das Handling der Navigation. Seien Sie sich auch im Klaren darüber, dass Anwender mit nicht DHTML-fähigen Browsern oder deaktiviertem JavaScript so wie das Beispiel oben notiert ist überhaupt keine Möglichkeit zur Navigation haben. Um zumindest Anwendern, die JavaScript deaktiviert haben, ein Navigieren zu ermöglichen, könnten Sie beispielsweise an einer geeigneten Stelle im normalen Text einen `noscript`-Bereich notieren, in dem Sie die Navigationsverweise noch mal notieren.

## 22.3 Laufende Datums-/Uhrzeitanzeige

### 22.3.1 Hinweise zu diesem Beispiel

Dieses Beispiel zeigt, wie Sie eine laufende Uhr, die Sie beliebig formatieren können, so realisieren können, dass sie ab Internet Explorer V4.x und Netscape V4.x funktioniert. In der Praxis eignet sich eine solche laufende Uhr z.B. für Seiten, die dauerhaft eingeblendet werden, z.B. in einem Frame-Fenster, dessen Inhalt sich beim Surfen nicht ändert, oder auf Seiten, die Multimedia-Spiele oder einen eingebetteten Chat enthalten – also auf Seiten, die der Anwender lange

anzeigt, ohne darin weiterzuscrollen oder die Seite zu verlassen. Das Beispiel benutzt die DHTML-Bibliothek, um das zu leisten, was in dem Beispiel an dynamischem HTML erforderlich ist.



22.3: Hier kommt die Uhr: Während der Surfer die Web-Seite durchliest, läuft die Uhrzeit weiter.

### 22.3.2 Quelltext mit Erläuterungen

Das Beispiel zeigt eine vollständige HTML-Datei, in der die laufende Datums-/Uhrzeitanzeige eingebettet ist. Die Datei *dhtml.js*, in der die Funktionen der DHTML-Bibliothek enthalten sind, muss vorhanden sein und sich im Beispiel im gleichen Verzeichnis befinden wie die HTML-Datei. Ein Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript" src="dhtml.js"></script>
<script type="text/javascript">
<!--
function ZeitAnzeigen() {
var Wochentagname = new
Array("Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag");

var Jetzt = new Date();
var Tag = Jetzt.getDate();
var Monat = Jetzt.getMonth() + 1;
var Jahr = Jetzt.getFullYear();
if(Jahr < 999) Jahr += 1900;
var Stunden = Jetzt.getHours();
var Minuten = Jetzt.getMinutes();
var Sekunden = Jetzt.getSeconds();
var WoTag = Jetzt.getDay();
var Vortag = ((Tag < 10) ? "0" : "");
var Vormon = ((Monat < 10) ? ".0" : ".");
var Vorstd = ((Stunden < 10) ? "0" : "");
var Vormin = ((Minuten < 10) ? ":0" : ":");
var Vorsek = ((Sekunden < 10) ? ":0" : ":");
var Datum = Vortag + Tag + Vormon + Monat + "." + Jahr;
var Uhrzeit = Vorstd + Stunden + Vormin + Minuten + Vorsek + Sekunden;
var Gesamt = Wochentagname[WoTag] + ", " + Datum + ", " + Uhrzeit;
```

```

if(DHTML) {
    if(NS) setCont("id","Uhr",null,"<span class=\"Uhr\">" + Gesamt + "</span>");
    else    setCont("id","Uhr",null,Gesamt);
}
else return;

window.setTimeout("ZeitAnzeigen()",1000);
}
//-->
</script>
<style type="text/css">
<!--
#Uhr { position:absolute; top:10px; left:10px; }
#RestDerSeite { position:absolute; top:50px; left:10px; }
.Uhr { font-family:Arial; font-size:24px; color:blue; }
.RestDerSeite { font-family:Arial; color:black; }
-->
</style>

</head>
<body onLoad="window.setTimeout('ZeitAnzeigen()',1000)">

<div id="Uhr" class="Uhr">&nbsp;</div>

<div id="RestDerSeite" class="RestDerSeite">

<h1>Willkommen auf dieser Seite</h1>
<p>hier gehts ganz normal weiter</p>

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

```

Im Körper der HTML-Datei im Beispiel werden insgesamt zwei `div`-Bereiche definiert. Beide Bereiche erhalten einen `id`-Namen und über den jeweiligen `id`-Namen im `style`-Bereich im Dateikopf entsprechende CSS-Eigenschaften zugewiesen. Beide Bereiche werden absolut positioniert. Ohne diesen Trick wäre die ganze Sache mit Netscape 4.x nicht realisierbar. Der erste Bereich dient zur Anzeige von Datum und Uhrzeit, und der zweite Bereich enthält den gesamten restlichen Inhalt der Datei, weshalb er auch den Namen `RestDerSeite` trägt.

Der erste Bereich enthält zunächst nur ein geschütztes Leerzeichen. So sieht man zunächst noch nichts von dem Bereich, aber das Leerzeichen ist wichtig, damit der Bereich einen Inhalt hat. Ansonsten würde beim ersten dynamischen Zugriff auf seinen Inhalt mit DOM-fähigen Browsern und den Funktionen der DHTML-Bibliothek ein Fehler erzeugt.

Die Beispieldatei enthält ferner zwei JavaScript-Bereiche im Dateikopf. Im ersten, leeren JavaScript-Bereich wird die DHTML-Bibliothek eingebunden, im zweiten steht die Funktion `ZeitAnzeigen()`, die für die Zeitanzeige verantwortlich ist. Im einleitenden `<body>`-Tag ist der Event-Handler `onLoad`, der eintritt, wenn die Datei und alle ihre eingebundenen Ressourcen

vollständig geladen sind. Diesem Event-Handler ist ein Funktionsaufruf zugeordnet. Er startet die Funktion `ZeitAnzeigen()`, jedoch eingebettet in eine `setTimeout`-Anweisung, die den Aufruf um 1 Sekunde bzw. 1000 Millisekunden verzögert. In der ersten Sekunde wird also noch nichts angezeigt.

Innerhalb der Funktion `ZeitAnzeigen()` wird zunächst ein Array für die deutschen Wochentage definiert. Anschließend ermittelt die Funktion mit Hilfe des `Date`-Objekts die aktuelle Zeit. Mit Hilfe diverser Formatierbefehle erzeugt sie dann eine Zeichenkette, die eine ordentliche Datums-/Uhrzeitdarstellung nach deutscher Konvention enthält. Diese Zeichenkette wird in der Variablen mit dem Namen `Gesamt` gespeichert. All diese Dinge haben noch nichts mit der Anzeige zu tun.

Der eigentliche DHTML-Teil ist dank der eingebundenen DHTML-Bibliothek kein großes Problem. Mit `if(DHTML)` wird zunächst abgefragt, ob überhaupt Dynamisches HTML verfügbar ist. Der gesamte Code zum dynamischen Schreiben in den `div`-Bereich wird davon abhängig gemacht. So werden Fehlermeldungen in Browsern vermieden, die zwar JavaScript, aber kein Dynamisches HTML beherrschen. Dann wird abgefragt, ob mit dem älteren Netscape-Modell gearbeitet wird. Wenn ja, sieht das dynamische Schreiben in den `div`-Bereich ein klein wenig anders aus als in den übrigen Fällen. Der Grund ist eine weitere Schwäche des Netscape 4-Browsers. Er würde die CSS-Klasse `Uhr` im einleitenden `div`-Tag ignorieren, wenn man sie ihm nicht noch mal in Form eines `span`-Elements mit in den dynamischen Bereich schreiben würde. Nur so erhält die Datums-/Uhrzeitanzeige die CSS-Formatierungen, die im Dateikopf im `style`-Bereich für die `Uhr`-Klasse festgelegt werden. Das dynamische Ändern des `div`-Inhalts geschieht durch Aufruf der Bibliotheksfunktion `setCont()`, die sich um alle weiteren Einzelheiten kümmert.

In der letzten Anweisung innerhalb der Funktion `ZeitAnzeigen()` ruft sich die Funktion selber wieder auf, und zwar wieder mit einer durch `window.setTimeout()` bewirkten Zeitverzögerung von 1000 Millisekunden oder einer Sekunde. Dadurch passiert eine Sekunde lang nichts, und dann wird die Zeit erneut geschrieben.

## 22.4 Bilderbuch zum Umblättern

### 22.4.1 Hinweise zu diesem Beispiel

Es gibt Seiten, auf denen Ihre Besucher Bilder sehen wollen. Wenn Sie beispielsweise über eine Stadt oder ein Land berichten oder über ein Treffen mit Freunden, mögen die Besucher gerne Bilder davon sehen. Das hier vorgestellte Beispiel zeigt, wie man solche Bilder einmal anders präsentieren kann. Ohne viel scrollen oder andere Seiten aufrufen zu müssen, kann der Anwender in den vorhandenen Bildern vor- und zurückblättern. Damit der Bildaufbau auch mal etwas anders aussieht als sonst, ist eine Art »Vorhang-Auf«-Effekt eingebaut. Die Bilder werden nicht einfach angezeigt, sondern »enthüllt«.



22.4: Das Bilderbuch benutzt Überblendeffekte zwischen den einzelnen Darstellungen.

#### 22.4.2 Quelltext mit Erläuterungen

Das Beispiel zeigt eine vollständige HTML-Datei mit dem Bilderbuch. Die Datei *dhtml.js*, in der die Funktionen der DHTML-Bibliothek enthalten sind, muss vorhanden sein und sich im Beispiel im gleichen Verzeichnis befinden wie die HTML-Datei. Das Beispiel:

```
<html><head><title>Test</title>
<script type="text/javascript" src="dhtml.js"></script>
<script type="text/javascript">
<!--
var Bild = new Array();
Bild[0] = new Image(); Bild[0].src="bilderbuch01.gif";
Bild[1] = new Image(); Bild[1].src="bilderbuch02.gif";
Bild[2] = new Image(); Bild[2].src="bilderbuch03.gif";
Bild[3] = new Image(); Bild[3].src="bilderbuch04.gif";
Bild[4] = new Image(); Bild[4].src="bilderbuch05.gif";

var zeige = 0;
var Breite = 300;
var aktuelleBreite = 0;
var Schrittweite = 6;
var Schrittverzoeigerung = 1;

function Blaettern(Richtung) {
    zeige = zeige + Richtung;
    if(zeige > Bild.length - 1) zeige = 0;
    else if(zeige < 0) zeige = Bild.length - 1;
    if(DOM || MS)
```



```

    getElem("id","Bildbereich",null).style.clip="rect(0 0 225 0)";
if(DOM && !MS && !OP) {
    var img = document.createElement("img");
    var src = document.createAttribute("src");
    src.nodeValue = Bild[zeige].src;
    img.setAttributeNode(src);
    getElem("id","Bildbereich",null).replaceChild(img,
getElem("id","Bildbereich",null).firstChild);
}
else if(MS) {
    document.all.Bildbereich.innerHTML = "<img src=\"\" + Bild[zeige].src + \"\">";
}
else if(NS) {
    getElem("index",0,null).visibility = "hide";
    setCont("index",0,null,"<img src=\"\" + Bild[zeige].src + \"\">");
    getElem("index",0,null).clip.right = 0;
    getElem("index",0,null).visibility = "show";
}
BildAufbauen();
}

function BildAufbauen() {
    if(aktuelleBreite <= Breite) {
        if(DOM || MS && !OP)
            getElem("id","Bildbereich",null).style.clip="rect(0 "+ aktuelleBreite +" 225
0)";
        else if(NS)
            getElem("index",0,null).clip.right = aktuelleBreite;
        aktuelleBreite = aktuelleBreite + Schrittweite;
        window.setTimeout("BildAufbauen()", Schrittverzoegerung);
    }
    else {
        aktuelleBreite = 0;
        if(DOM || MS)
            getElem("id","Bildbereich",null).style.clip="rect(0 "+ Breite +" 225 0)";
        else if(document.layers)
            getElem("index",0,null).clip.right = Breite;
    }
}

function SEITE_init() {
    if(DOM && !MS && !OP) {
        img = document.createElement("img");
        var src = document.createAttribute("src");
        src.nodeValue = Bild[0].src;
        img.setAttributeNode(src);
        getElem("id","Bildbereich",null).appendChild(img);
    }
}

```

```

}
else if(document.all)
    document.all.Bildbereich.innerHTML = "<img src=\"\" + Bild[0].src + \"\">";
else if(NS)
    setCont("index",0,null,"<img src=\"\" + Bild[0].src + \"\">");
}
//-->
</script>
<style type="text/css">
<!--
#Bildbereich {
    position:absolute; left:10px; top:140px; padding:0px; clip:rect(0 300 225 0);
visibility:show; }
#Navigationsbereich {
position:absolute; left:320; top:140; width:100; font-size:18pt; }
-->
</style>

</head>
<body bgcolor="#FFFFFF" onLoad="SEITE_init()">

<div id="Bildbereich">

</div>

<div id="Navigationsbereich">
[<a href="javascript:Blattetern(-1)"><b><<</b></a>]
[<a href="javascript:Blattetern(1)"><b>>></b></a>]
</div>

</body>
</html>

```

Im Körper der HTML-Datei des Beispiels werden zwei `div`-Bereiche definiert. Der erste `<div>`-Bereich ist zum Anzeigen der jeweils aktuellen Grafik gedacht. Der zweite Bereich enthält Verweise, mit deren Hilfe der Anwender vorwärts und rückwärts zwischen den Grafiken blättern kann. Beide Bereiche werden mit Hilfe entsprechender CSS-Eigenschaften absolut positioniert. Die entsprechenden CSS-Definitionen stehen im Dateikopf im `style`-Bereich und beziehen sich auf die `id`-Namen der Bereiche.

Die Beispieldatei enthält ferner zwei JavaScript-Bereiche im Dateikopf. Im ersten, leeren JavaScript-Bereich wird die DHTML-Bibliothek eingebunden. Im einleitenden `<body>`-Tag wird wie üblich die Funktion `DHTML_init()` aus der eingebundenen DHTML-Bibliothek aufgerufen, um diese zu initialisieren. Ferner wird aber auch die Funktion `SEITE_init()` aufgerufen, die im JavaScript im Dateikopf notiert ist. Diese Funktion sorgt dafür, dass das erste Bild angezeigt wird.

Damit das Beispiel des Bilderbuches zum Umblättern sauber funktioniert, passiert jedoch schon vorher etwas. Gleich beim Einlesen der Datei, also außerhalb jeder Funktion, wird der JavaScript-Code ausgeführt, der zu Beginn des zweiten JavaScript-Bereichs notiert ist. Dort werden

erst mal alle Grafiken eingelesen, die im Verlauf des Umblätterns angezeigt werden sollen. Denn nur wenn die Grafiken zu diesem Zeitpunkt bereits im Arbeitsspeicher sind, gibt es keine Synchronisationsprobleme. Dazu kommt das `images`-Objekt von JavaScript zum Einsatz. Zuerst wird ein neuer, leerer Array angelegt. Dann wird der Array mit Grafikobjekten gefüllt. Dem jeweiligen Image-Objekt, das zu diesem Zweck erzeugt wird, wird über die `src`-Eigenschaft eine gewünschte Grafikdatei zugewiesen. Anschließend werden noch ein paar globale Variablen definiert, die später benötigt werden.

Die Funktion `SEITE_init()` zeigt dann nach ihrem Aufruf die erste Grafik innerhalb des vorgesehenen Anzeigebereichs an. Dabei ist mal wieder viel Unterscheidungsarbeit nötig, um die verschiedenen Browser und ihre DHTML-Modelle zu bedienen. Das DOM-Modell ist in diesem Fall für den Internet Explorer vorläufig noch nicht anwendbar, da dieser zumindest in der Version 5.x das Erzeugen von Attributknoten nach DOM-Syntax noch nicht unterstützt. Mit der Abfrage `if(DOM && !MS && !OP)` werden der Internet-Explorer und auch Opera von der Bearbeitung nach dem DOM-Modell ausgeklammert. Nur Netscape 6.x gelangt in diesen Bereich. Dort wird mit den Funktionen `createElement()` und `createAttribute()` des `document`-Objekts der HTML-Code für das neue Bild erzeugt und dann zusammengebastelt. Der Internet Explorer gelangt dagegen in den Zweig, der mit `else if(MS)` beginnt. Dort wird sich der praktischen Methode `innerHTML` des `all`-Objekts bedient, um den gewünschten HTML-Code zusammenzubasteln. Schließlich bleibt noch Netscape 4.x übrig, der in den Zweig `else if(NS)` gelangt und die Grafik mit der Funktion `setCont()` aus der DHTML-Bibliothek zusammengesetzt bekommt. An diesem Beispiel sehen Sie deutlich, wie mühsam das Unterscheiden zwischen Browsern und DHTML-Modellen oft ist. Außerdem können Sie sehen, dass das DOM-Modell bei allen Vorteilen manchmal auch sehr umständlich zu handhaben ist. Eine Methode wie `innerHTML` beim `all`-Objekt gibt es im DOM nicht, weil dort neue Knoten streng nach den Regeln der Auszeichnungssprachen-Syntax erstellt und zusammengestellt werden müssen. Opera ist gegenwärtig noch nicht in der Lage, Inhalte dynamisch zu ändern, und wird komplett ausgeschlossen.

Das Blättern zwischen einzelnen Grafiken wird durch die beiden Verweise im zweiten `div`-Bereich angestoßen. Beide Verweise rufen die gleiche Funktion namens `Blaettern()` auf, jedoch mit unterschiedlicher Parameterübergabe. Der Vorwärtsverweis übergibt 1, der Rückwärtsverweis -1. Dadurch kann die Funktion `Blaettern()` diesen Parameter gleich zur Verwaltung benutzen.

Die Funktion `Blaettern()` fragt zunächst ab, ob entweder das erste oder das letzte Bild der Serie erreicht ist, und legt im entsprechenden Fall wieder das letzte bzw. erste Bild als das anzuzeigende fest. So entsteht ein Endlos-Effekt für den Anwender beim Blättern. Anschließend wird der Inhalt des Anzeigebereichs für das Bild neu geschrieben, und zwar mit dem neuen Bild. Dabei muss wieder genauso mühsam zwischen den Fähigkeiten der Browser unterschieden werden wie in der Funktion `SEITE_init()`. Am Ende wird dann noch die Funktion `BildAufbauen()` aufgerufen.

Die Funktion `BildAufbauen()` ist für den dynamischen Aufroll-Effekt beim Anzeigen der neuen Grafik zuständig. Dazu wird von der CSS-Eigenschaft `clip` Gebrauch gemacht. Um sie zu setzen, kann die Basisfunktion `getElement()` aus der DHTML-Bibliothek für den Elementzugriff verwendet werden. Allerdings ist der Aufruf für Netscape 4.x anders als bei den anderen Browsern, weshalb wieder eine `if-else`-Weiche nötig ist. Nach Durchlaufen der Wertveränderung des rechten `Clip-Rands` ruft sich die Funktion `BildAufbauen()` selbst wieder auf, und zwar

innerhalb von `window.setTimeout()`. Dabei kommen die Variablen `Schrittweite` und `Schrittverzögerung` zum Einsatz, die zu Beginn des Script-Bereichs notiert sind. Probieren Sie dort bei der anfänglichen Wertzuweisung ruhig auch mal andere Werte aus.

Am Ende, wenn die oberste `if`-Bedingung der Funktion `BildAufbauen()` erfüllt ist, nämlich dass die aktuelle Breite größer als die Anzeigebreite der Grafik ist, gelangt die Funktion in den `else`-Zweig. Dort wird der `clip`-Wert für rechts auf den Endwert gesetzt, und die Prozedur des Umblätterns ist abgeschlossen.

## 23 Internationalisierung

### 23.1 Computer und geschriebene Sprache

#### 23.1.1 Bits, Bytes und Zeichen

Die beiden Grundeinheiten in jedem heutigen Computer sind die Einheiten **Bit** und **Byte**. Ein Byte ist bei den heute üblichen Systemen als Folge von 8 Bit definiert (man spricht auch von **Octets**). Da jedes Bit zwei Zustände haben kann, nämlich 0 oder 1 bzw. ja oder nein, lassen sich mit einer Folge von 8 Bit genau  $256 (= 2^8)$  unterschiedliche Zustände realisieren. Ein Byte kann also 256 unterschiedliche Werte haben. Da im Computer immer auch die 0 dazugehört, können in einem Byte dezimal ausgedrückt Werte zwischen 0 und 255 stehen.

Wenn ein laufendes Programm im Computer eine Datei zeichenweise in den Arbeitsspeicher einliest, stehen im Arbeitsspeicher anschließend nur Byte-Werte. Von Zeichen unseres Alphabets ist auf dieser Ebene noch keine Rede. Damit aus den Byte-Werten lesbare Zeichen werden, die sich am Bildschirm darstellen lassen, braucht es eine Konvention, die festlegt, welcher Byte-Wert als welches Zeichen dargestellt werden soll. Diese Aufgabe haben die so genannten **Zeichensätze**. Zeichensätze sind Tabellen, die einem Byte-Wert ein Zeichen zuordnen, das in unseren Schriftkulturen eine Bedeutung hat.

Die Zeichensätze sind EDV-historisch gewachsene Gebilde. Bis zum Aufkommen der Personalcomputer benutzten viele Rechner noch 7 Bit lange Grundeinheiten, mit denen sich nur 128 unterschiedliche Zustände darstellen lassen. Noch früher waren es auch mal 6 und 5 Bit lange Grundeinheiten. Auf der 7 Bit langen Grundeinheit beruhten die ersten Zeichensätze, die historisch den Durchbruch schafften: der ASCII-Zeichensatz und der EBCDIC-Zeichensatz. Dabei setzte sich vor allem der ASCII-Zeichensatz durch, weil er im erfolgreichen Unix-Betriebssystem und in den aufkommenden Personalcomputern zum Einsatz kam.

Beim ASCII-Zeichensatz sind die ersten 32 Zeichen für Steuerzeichen reserviert, etwa für Tastatur-Impulse wie Zeilenumbruch. Die Zeichen zwischen 32 und 127 sind darstellbare Zeichen, darunter alle Ziffern, Satzzeichen und Buchstaben, die ein Amerikaner so braucht (denn der ASCII-Zeichensatz kommt natürlich aus den USA).

Lange Zeit war ASCII der einzige verbreitete Standard. Da die neueren Computer aber 256 lange Bytes hatten, war es folgerichtig, für die Werte zwischen 128 und 255 neue Verwendungszwecke zu finden. Dabei entwickelten sich jedoch proprietäre Lösungen. MS DOS beispielsweise benutzt einen »erweiterten ASCII-Zeichensatz«, der aber nicht viel mehr ist als eine schöne Umschreibung für die Microsoft-eigene Belegung der Zeichen 128 bis 255 speziell für die Bedürfnisse von MS-DOS.

Um auch hierfür einen Standard zu schaffen, ersann die amerikanische Standardisierungs-Organisation den ANSI-Zeichensatz. Dieser Zeichensatz übernimmt für die Zeichen 0 bis 127 den ASCII-Zeichensatz und definiert für die Werte zwischen 128 und 255 etliche Sonderzeichen,

darunter wichtige Alphabetzeichen verbreiteter Sprachen, etwa deutsche Umlaute, französische Accent-Zeichen oder spanische Zeichen mit Tilde. Dazu kamen diverse verbreitete kaufmännisch-wissenschaftliche Zeichen.

Der Bedarf an international gültigen Zeichensätzen wird jedoch immer größer. Ein Versuch, ein solches Set von Zeichensätzen zu etablieren, stellt die iso-8859-Familie dar (siehe Kapitel 23.2).

### 23.1.2 Beispiel für Zeichensätze

Ein Beispiel soll das Prinzip der Zeichensätze verdeutlichen. Die folgende Abbildung zeigt zwei Zeichensätze: den MS-DOS-Zeichensatz und den ANSI-Zeichensatz, den z.B. MS Windows als Voreinstellung verwendet.

| DOS-Zeichensatz                                           |   |   |    |   |   |   |   |   |   |   |   |   |   |   | ANSI-Zeichensatz                                 |   |   |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-----------------------------------------------------------|---|---|----|---|---|---|---|---|---|---|---|---|---|---|--------------------------------------------------|---|---|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| !                                                         | " | # | \$ | % | & | ' | ( | ) | * | + | , | - | . | / | !                                                | " | # | \$ | % | & | ' | ( | ) | * | + | , | - | . | / |   |   |   |   |
| 0                                                         | 1 | 2 | 3  | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | 0                                                | 1 | 2 | 3  | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |   |   |   |
| @                                                         | A | B | C  | D | E | F | G | H | I | J | K | L | M | N | O                                                | @ | A | B  | C | D | E | F | G | H | I | J | K | L | M | N | O |   |   |
| P                                                         | Q | R | S  | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _                                                | P | Q | R  | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |   |   |
| `                                                         | a | b | c  | d | e | f | g | h | i | j | k | l | m | n | o                                                | ` | a | b  | c | d | e | f | g | h | i | j | k | l | m | n | o |   |   |
| p                                                         | q | r | s  | t | u | v | w | x | y | z | { |   | } | ~ |                                                  | p | q | r  | s | t | u | v | w | x | y | z | { |   | } | ~ |   |   |   |
|                                                           |   |   |    |   |   |   |   |   |   |   |   |   |   |   |                                                  |   |   |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Ç                                                         | ü | é | â  | ä | à | å | ç | ê | ë | è | ì | í | î | ï | Ä                                                | Å |   | ,  | f | " | " | " | " | " | " | " | " | " | " | " | " |   |   |
| É                                                         | æ | Æ | ö  | ø | ù | û | ý | ÿ | Ö | Ü | Ø | £ | ¥ | × | ƒ                                                |   |   | '  | " | " | " | " | " | " | " | " | " | " | " | " | " |   |   |
| á                                                         | í | ó | ú  | ñ | Ñ | ª | º | ¿ | ® | ™ | ½ | ¾ | ¿ | « | »                                                |   |   | ı  | ¢ | £ | ¤ | ¥ | ¦ | § | ¨ | © | ª | « | ¬ | ® | ¯ |   |   |
| Ł                                                         | ł | ł | ł  | ł | ł | ł | ł | ł | ł | ł | ł | ł | ł | ł | ł                                                |   |   | ±  | ² | ³ | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ | ¾ | ¿ |   |
| Š                                                         | š | Š | š  | Š | š | Š | š | Š | š | Š | š | Š | š | Š | š                                                |   |   | À  | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î |   |
| Œ                                                         | œ | Œ | œ  | Œ | œ | Œ | œ | Œ | œ | Œ | œ | Œ | œ | Œ | œ                                                |   |   | Đ  | Ñ | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û | Ü | Ý | Þ |   |
| Ó                                                         | ß | Ö | ö  | Ö | ö | µ | þ | þ | Ú | Û | Ü | Ý | Ý | Ý | Ý                                                |   |   | à  | á | â | ã | ä | å | æ | þ | è | é | ê | ë | ì | í | î | ï |
| -                                                         | ± | = | ¾  | ¶ | § | ÷ | , | ° | · | . | 1 | 2 | 3 | 4 | 5                                                |   |   | ö  | ñ | ò | ó | ô | õ | ÷ | ø | ù | ú | û | ü | ý | þ | ÿ |   |
| ASCII-Teil des Zeichensatzes<br>DOS-Zeichensatz erweitert |   |   |    |   |   |   |   |   |   |   |   |   |   |   | ASCII-Teil des Zeichensatzes<br>ANSI-Zeichensatz |   |   |    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

#### 23.1: DOS- und ANSI-Zeichensatz im Vergleich

Der Abbildung können Sie entnehmen, dass die oberen Teile (dunkler) beider Zeichensätze identisch sind, da beide Zeichensätze den ASCII-Zeichensatz für die ersten 128 Zeichen übernehmen (die ersten 32 Zeichen fehlen in der Abbildung, da es sich bei diesen Zeichen um Steuerzeichen handelt, die nicht abbildbar sind). Die unteren Teile (heller) sind dagegen unterschiedlich. Wo im ANSI-Zeichensatz beispielsweise ein kleines deutsches ü liegt, bietet der erweiterte DOS-Zeichensatz eine hochgestellte 3 an.

Wenn Sie mit MS Windows arbeiten, können Sie das selbst ausprobieren: Erzeugen Sie unter MS Windows mit einem ANSI-Texteditor (z.B. Notepad) eine leere Datei: tippen Sie einfach so etwas wie »äääööööüüüü« ein. Die Eingaben speichern Sie unter einem Dateinamen ab. Anschließend rufen Sie die DOS-Eingabeaufforderung auf und geben `edit` ein. Dann öffnet sich der DOS-Editor. Damit öffnen Sie die zuvor unter Windows abgespeicherte Datei. Nun können Sie sehen, was der Zusammenhang zwischen Byte und Zeichensatz ist: Es werden zwar genauso

viele Zeichen angezeigt wie unter MS Windows, aber es sind andere Zeichen. Der Grund ist, dass im Arbeitsspeicher immer nur Byte-Werte stehen. Was ein Programm jedoch daraus macht, hängt davon ab, welchen Zeichensatz es benutzt.

Das Beispiel zwischen dem MS-DOS-Zeichensatz und dem ANSI-Zeichensatz wurde hier bewusst ausgewählt, da es sich an ein und demselben Rechner testen lässt. Das Beispiel soll aber vor allem auch dazu ermuntern, tiefer in die Geheimnisse der Zeichensätze einzusteigen und sich darüber bewusst zu werden, dass nichts von dem, was im Anzeigefenster eines Web-Browsers zu sehen ist, selbstverständlich ist, sondern dass alles auf Konventionen beruht, die computertechnisch gesehen zum Teil auf wesentlich tieferen Ebenen aufsetzen als HTML und andere standardisierte Web-Sprachen.

### 23.1.3 Die Grenzen von Bytes und Zeichensätzen sprengen

Zeichensätze decken einzelne Schriftkulturen und damit verbundene Sprachen oder Sprachfamilien ab. Problematisch wird es, wenn mehrsprachige Dokumente erstellt werden sollen, die Schriften ganz unterschiedlicher Zeichensätze enthalten. Auch für nichtalphabetische Schriftkulturen sind Zeichensätze ungeeignet. In unseren Zeiten der Globalisierung wird es jedoch immer wichtiger, für solche Probleme eine standardisierte EDV-technische Lösung zu finden, die sich auf den verschiedensten Computersystemen durchsetzt. Eine solche Lösung gibt es auch schon: das Unicode-System (Kapitel 23.3). Unicode soll mittelfristig das heute übliche System der Zeichensätze ablösen. Derzeit geschieht auch schon viel in dieser Richtung. Neuere Betriebssysteme bieten meist schon Schriftarten an, die nicht mehr auf einen Zeichensatz beschränkt sind, sondern das komplette Unicode-Zeichensystem oder zumindest große Teile davon abdecken. Auch die meisten modernen Anwendungen speichern Texte nicht mehr mit einem Byte pro Zeichen, sondern mit zwei Byte pro Zeichen. Damit ist die Grundlage zur Unterstützung des Unicode-Systems gegeben.

### 23.1.4 Die Bedeutung der Schriftarten

Schriftarten sind Beschreibungsmodelle, um auf Ausgabemedien wie Bildschirm oder Drucker Zeichen abzubilden. Jedes heute übliche Betriebssystem enthält so genannte **Systemschriften**. Das sind Schriftarten, die auf jeden Fall genau die Zeichen enthalten, die in dem Zeichensatz definiert sind, auf dem das Betriebssystem per Voreinstellung basiert. Unter MS Windows gibt es beispielsweise eine solche Schriftart namens `System`. Daneben gibt es auf modernen Rechnern definierte Schnittstellen für beliebige Schriftarten. Verbreitet ist z.B. die Adobe-Schnittstelle für Schriftarten (PostScript). Unter MS Windows kommt eine eigene Schnittstelle hinzu (TrueType).

Solche Schriftarten können auf die zur Verfügung stehenden Byte-Werte beliebige Darstellungsmuster legen. So gibt es auch Schriftarten wie `WingDings` oder `ZapfDingbats`, die fast nur Symbole und Icons enthalten. Wichtig sind jedoch vor allem auch Schriftarten, die zwar ansprechend aussehen, aber gleichzeitig einen bestimmten Zeichensatz unterstützen, d.h. alle Zeichen dieses Zeichensatzes darstellen, und zwar genau auf den Byte-Werten, die im Zeichensatz dafür vorgesehen sind. Nur durch solche Schriftarten wird es möglich, festgelegte Zeichensätze in eine grafisch darstellbare Form zu bringen. Die folgende Abbildung zeigt ein Beispiel für diesen Zusammenhang.



23.2: Zusammenhang zwischen Schriftart und Zeichensatz.

Bei modernen Schriftarten, die das ganze Unicode-System abdecken, ist das Prinzip das gleiche. Nur mit dem Unterschied, dass dort nicht nur 256 Zeichen adressiert werden können, sondern derzeit bis zu 65536 Zeichen, künftig noch viel mehr, aus dem Grund, weil dort zwei oder gar vier Bytes für Zeichenwerte verwendet werden, wodurch viel höhere Werte als 256 möglich sind.

### 23.1.5 Schriftkulturen mit anderer Schreibrichtung

Da die Computerindustrie historisch gesehen in den USA und Europa entstand, bauen die heutigen Hardwaresysteme und Betriebssysteme auf Prinzipien auf, die zunächst als selbstverständlich galten. Wenn Sie in einem Textverarbeitungsprogramm einen Text tippen, wandert der Cursor beim Schreiben von links nach rechts. Automatische Zeilenumbrüche erfolgen nach typischen Begrenzerzeichen westlicher Sprachen wie Leerzeichen oder Silbentrennstrich.

Es gibt jedoch etliche Schriftkulturen, die eine andere Schreibrichtung als die unsrige haben. Dazu gehören etwa die arabische Schrift, die hebräische Schrift oder die fernöstlichen Schriftkulturen. Um solche Schriftkulturen auf Computern abzubilden, sind zusätzliche Fähigkeiten der Software erforderlich. Denn es gilt nicht nur, die Schriftelemente abzubilden, sondern auch die Editierichtung bei der Texteingabe und die Ausgaberrichtung auf Medien wie Bildschirm oder Drucker an die Schreibrichtung der entsprechenden Schriftkultur anzupassen.

In HTML gibt es deswegen Sprachbestandteile wie das Universalattribut `dir=` oder das Element `bdo`, mit deren Hilfe Web-Autoren Angaben zur Schreibrichtung machen können. Die softwareseitige Umsetzung »on the fly« ohne das Erfordernis, zusätzliche Spezialsoftware installieren zu müssen, funktioniert bei neueren Browsern auch schon zum Teil.

## 23.2 Zeichensätze (ISO-8859-Familie und andere)

### 23.2.1 Die ISO-8859-Zeichensätze

Die ISO-8859-Familie wurde vom European Computer Manufacturer's Association (ECMA) entwickelt. Es handelt sich um ein Set von standardisierten Zeichensätzen für alphabetische Schriften. Dazu gehören die lateinischen Schriften, auf denen die meisten Sprachen Westeuropas und Amerikas beruhen, oder etwa die kyrillischen Schriften.



Alle Zeichensätze dieser Familie basieren auf der 1-Byte-Abbildung eines Zeichens. Das heißt, alle Zeichensätze enthalten 256 mögliche Zeichen. Bei allen Zeichensätzen sind die ersten 128 Zeichen, also die Zeichen mit den Werten 0 bis 127, identisch mit dem ASCII-Zeichensatz. Das hat den Vorteil, dass die üblichen lateinischen Groß- und Kleinbuchstaben, die arabischen Ziffern und die üblichen Sonderzeichen wie Satzzeichen oder kaufmännische Zeichen in all diesen Zeichensätzen immer zur Verfügung stehen.

| ASCII-Zeichensatz |   |   |   |   |   |    |   |   |   |   |
|-------------------|---|---|---|---|---|----|---|---|---|---|
| +                 | 0 | 1 | 2 | 3 | 4 | 5  | 6 | 7 | 8 | 9 |
| 30                |   |   | ! | " | # | \$ | % | & | ' |   |
| 40                | ( | ) | * | + | , | -  | . | / | 0 | 1 |
| 50                | 2 | 3 | 4 | 5 | 6 | 7  | 8 | 9 | : | ; |
| 60                | < | = | > | ? | @ | A  | B | C | D | E |
| 70                | F | G | H | I | J | K  | L | M | N | O |
| 80                | P | Q | R | S | T | U  | V | W | X | Y |
| 90                | Z | [ | \ | ] | ^ | _  | ` | a | b | c |
| 100               | d | e | f | g | h | i  | j | k | l | m |
| 110               | n | o | p | q | r | s  | t | u | v | w |
| 120               | x | y | z | { |   | }  | ~ |   |   |   |

23.3: ASCII-Zeichensatz

Oben ist der ASCII-Zeichensatz abgebildet. Die darstellbaren Zeichen dieses Zeichensatzes haben die Werte zwischen 32 und 127. Wert 32 ist das Leerzeichen. Alle ISO-8859-Zeichensätze enthalten diesen Zeichensatz.

### ***Zur Lesart dieser und der folgenden Zeichensatz-Tabellen***

Den Wert eines Zeichens ermitteln Sie, indem Sie die Werte aus Spalten- und Zeilenüberschrift des Zeichens addieren. Ein großes K hat in dieser Tabelle beispielsweise den Zeilenwert 70 und den Spaltenwert 5. Daraus ergibt sich der Wert 75 für dieses Zeichen. Zeichenwerte können bei der Verwendung von Zeichensätzen und Unicodes in HTML eine Rolle spielen. Falls Sie nicht den Dezimalwert eines Zeichens benötigen, sondern den Hexadezimalwert, können Sie den Dezimal-Hexadezimal-Umrechner auf der Buch-CD benutzen (als Teil von SELFHTML).

So wie im ASCII-Zeichensatz die ersten 32 Zeichen (Zeichenwerte 0 bis 31) für Steuerzeichen reserviert sind, sparen auch die erweiterten Zeichensätze der ISO-8859-Familie einen Bereich von Zeichen aus. Es handelt sich um die ersten 32 Zeichen oberhalb des ASCII-Bereichs, also um die Zeichenwerte 128 bis 159. Die Tabellen zu den einzelnen Zeichensätzen weiter unten bilden daher die verbleibenden Zeichenwerte 160 bis 255 ab. Das Zeichen mit dem Wert 160 ist in all diesen Zeichensätzen ein erzwungenes Leerzeichen.

Viele der Zeichensätze der ISO-8859-Familie überlappen sich. Das ist Absicht. Die Zeichensätze sind so optimiert, dass sich mit je einem Zeichensatz möglichst viele Schriften möglichst vollständig abdecken lassen. Zeichen, die in mehreren Zeichensätzen der ISO-8859-Familie vorkommen, haben in der Regel immer den gleichen Wert, d.h. sie befinden sich in den unterschiedlichen Zeichensätzen an der gleichen Stelle.

Sechs der ISO-8859-Zeichensätze beziehen sich auf Schriften, die im Kern auf der lateinischen Schrift basieren. Diese Zeichensätze haben daher noch die Beinamen Latin-1 bis Latin-6.

### 23.2.2 Liste der ISO-8859-Zeichensätze

Die folgenden Tabellen zeigen die einzelnen Zeichensätze der ISO-8859-Familie. Wie Sie diese Zeichensätze in HTML einsetzen können, wird im Abschnitt 4.1.3 *Zeichensatzangabe als Meta-Angabe* beschrieben. Die blau dargestellten Namen sind jene Bezeichnungen, die Sie in HTML angeben können.

ISO-8859-1 (Latin-1)

Dieser Zeichensatz enthält die schriftspezifischen Zeichen für westeuropäische und amerikanische Sprachen. Der Zeichensatz deckt die Sprachen Albanisch, Dänisch, Deutsch, Englisch, Färöisch, Finnisch, Französisch, Galizisch, Irisch, Isländisch, Italienisch, Katalanisch, Niederländisch, Norwegisch, Portugiesisch, Schwedisch und Spanisch ab. Lediglich einzelne Zeichen wie das niederländische »ij« oder die deutschen Anführungszeichen unten fehlen.

| iso-8859-1 |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|
| +          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 160        |   | ı | Ɔ | £ | ¤ | ¥ | ¦ | § | ¨ | © |
| 170        | ª | « | ¬ | – | ® | ¯ | ° | ± | ² | ³ |
| 180        | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ |
| 190        | ¾ | ¿ | À | Á | Â | Ã | Ä | Å | Æ | Ç |
| 200        | È | É | Ê | Ë | Ì | Í | Î | Ï | Ð | Ñ |
| 210        | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û |
| 220        | Ü | Ý | Þ | ß | à | á | â | ã | ä | å |
| 230        | æ | ç | è | é | ê | ë | ì | í | î | ï |
| 240        | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù |
| 250        | ú | û | ü | ý | þ | ÿ |   |   |   |   |

ISO-8859-2 (Latin-2)

Dieser Zeichensatz enthält die schriftspezifischen Zeichen für die meisten mitteleuropäischen und slawischen Sprachen. Der Zeichensatz deckt die Sprachen Kroatisch, Polnisch, Rumänisch, Slowakisch, Slowenisch, Tschechisch und Ungarisch ab.

[illegible]

## ISO-8859-3 (Latin-3)

Dieser Zeichensatz deckt die Sprachen Esperanto, Galizisch, Maltesisch und Türkisch ab.

| iso-8859-3 |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|
| +          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 160        |   | Ħ | ˘ | £ | ¤ |   | Ĥ | ŝ | ˆ | İ |
| 170        | Ŗ | Ġ | Ĵ | - |   | Ž | ° | ħ | ² | ³ |
| 180        | ˘ | μ | ĥ | • | ˙ | ı | ŝ | ĝ | ĵ | ¥ |
| 190        |   | ž | Ā | Ā | Ā |   | Ä | Č | Č | Č |
| 200        | Ě | Ě | Ě | Ě | Ě | Ě | Ě |   |   | Ñ |
| 210        | Ō | Ō | Ō | Ō | Ō | × | Ō | Ū | Ū | Ū |
| 220        | Ū | Ū | Ŝ | ß | ā | ā | ā |   | ä | č |
| 230        | Ĉ | Ĉ | ĉ | ĉ | ĉ | ĉ | ĭ | ĭ | ĭ | ĭ |
| 240        |   | ñ | õ | ó | ô | ğ | ö | ÷ | ğ | ù |
| 250        | ú | ú | ü | ü | š | ˙ |   |   |   |   |

## ISO-8859-4 (Latin-4)

Dieser Zeichensatz enthält einige Zeichen der Sprachen Estnisch, Lettisch und Litauisch. Vergleichen Sie diesen Zeichensatz auch mit ISO 8859-10, der sehr ähnlich ist.

| iso-8859-4 |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|
| +          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 160        |   | Ā | Ķ | Ķ | × | Ī | Ļ | š | ˆ | š |
| 170        | Ē | Ģ | Ŧ | - | Ž | - | ° | q | ˘ | ŗ |
| 180        | ˘ | ī | Ķ | ˘ | š | ē | ģ | ķ | ņ |   |
| 190        | ž | ņ | Ā | Ā | Ā | Ā | Ā | Ā | Ē | Ē |
| 200        | Č | Ē | Ē | Ē | Ē | Ī | Ī | Ī | Đ | Đ |
| 210        | ō | Ķ | ô | õ | ö | × | ø | ų | ú | û |
| 220        | ü | ũ | ū | ß | ā | ā | ā | ā | ä | å |
| 230        | æ | ı | č | ē | ē | ë | è | í | î | ï |
| 240        | đ | ŗ | ō | ķ | ô | õ | ö | ÷ | ø | ų |
| 250        | ú | û | ü | ũ | ū | . |   |   |   |   |

## ISO-8859-5

Dieser Zeichensatz enthält kyrillische Zeichen. Er deckt weitgehend die Sprachen Bulgarisch, Macedonisch, Russisch, Serbisch und Ukrainisch ab.

| iso-8859-5 |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|
| +          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 160        |   | Ё | ѐ | ђ | є | ѕ | І | Ї | Ј | Љ |
| 170        | ъ | ѡ | Ѣ | – | Ѥ | Ѧ | Ѣ | Ѧ | Ѧ | Ѧ |
| 180        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 190        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 200        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 210        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 220        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 230        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 240        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |
| 250        | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ | Ѧ |   |   |   |   |



**ISO-8859-9 (Latin-5)**

Dieser Zeichensatz ist speziell für Türkisch gedacht. Er basiert auf ISO 8859-1, enthält jedoch anstelle der isländischen Sonderzeichen türkische Zeichen.

| iso-8859-9 |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|
| +          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 160        |   | ı | İ | £ | ¤ | ¥ | ¦ | § | ¨ | © |
| 170        | ª | « | ¬ | ­ | ® | ¯ | ° | ± | ² | ³ |
| 180        | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ |
| 190        | ¾ | ¿ | À | Á | Â | Ã | Ä | Å | Æ | Ç |
| 200        | È | É | Ê | Ë | Ì | Í | Î | Ï | Ğ | Ñ |
| 210        | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û |
| 220        | Ü | Ý | Ş | ß | à | á | â | ã | ä | å |
| 230        | æ | ç | è | é | ê | ë | ì | í | î | ï |
| 240        | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù |
| 250        | ú | û | ü | ı | ş | ý |   |   |   |   |

**ISO-8859-10 (Latin-6)**

Dieser Zeichensatz enthält speziell Zeichen für die Sprachen Grönländisch (Inuit) und Lappisch (Sami).

| iso-8859-10 |    |    |    |    |    |    |    |    |    |    |
|-------------|----|----|----|----|----|----|----|----|----|----|
| +           | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
| 160         |    | À  | É  | Ê  | Ë  | Ì  | Í  | Î  | Ï  | Ð  |
| 170         | Š  | Ŧ  | Ž  | –  | Ů  | Ű  | °  | ª  | «  | »  |
| 180         | ı  | ı̇ | ı̈ | ·  | ı̄ | ı̅ | ı̆ | ı̇ | ı̈ | ı̉ |
| 190         | ı̊ | ı̋ | ı̌ | ı̍ | ı̎ | ı̏ | ı̐ | ı̑ | ı̒ | ı̓ |
| 200         | ı̔ | ı̕ | ı̖ | ı̗ | ı̘ | ı̙ | ı̚ | ı̛ | ı̜ | ı̝ |
| 210         | ı̞ | ı̟ | ı̠ | ı̡ | ı̢ | ı̣ | ı̤ | ı̥ | ı̦ | ı̧ |
| 220         | ı̨ | ı̩ | ı̪ | ı̫ | ı̬ | ı̭ | ı̮ | ı̯ | ı̰ | ı̱ |
| 230         | ı̲ | ı̳ | ı̴ | ı̵ | ı̶ | ı̷ | ı̸ | ı̹ | ı̺ | ı̻ |
| 240         | ı̼ | ı̽ | ı̾ | ı̿ | ı̺ | ı̻ | ı̼ | ı̽ | ı̾ | ı̿ |
| 250         | ı̺ | ı̻ | ı̼ | ı̽ | ı̾ | ı̿ |    |    |    |    |

**23.2.3 Alle offiziellen Zeichensätze**

Es gibt etliche andere, zum Teil schon ältere Zeichensätze, die Konventionen für einzelne Schriften oder Schriftfamilien definieren, häufig landesspezifisch. Auf der folgenden Internet-Adresse werden die Namen dieser Zeichensätze gesammelt:

**[www.iana.org/assignments/character-sets](http://www.iana.org/assignments/character-sets)**

Diejenigen Zeichensatznamen, die dort in der Liste jeweils hinter Name: oder Alias: aufgelistet sind, können Sie in HTML angeben, wenn Sie einen bestimmten Zeichensatz erzwingen wollen (bevorzugt sollten jene Namen verwendet werden, die in der Liste mit dem Zusatz preferred MIME name ausgezeichnet sind). Möglich ist das z.B. dokumentweit mit der Meta-Angabe für den Default-Zeichensatz (Kapitel 4.1.3).

## 23.3 Das Unicode-System

### 23.3.1 Allgemeines zu Unicode

Unicode ist ein System, in dem die Zeichen oder Elemente aller bekannten Schriftkulturen und Zeichensysteme festgehalten werden. Durch dieses System wird es möglich, einem Computer »weltweit« zu sagen, welches Zeichen man dargestellt bekommen will. Voraussetzung ist natürlich, dass der Computer bzw. das ausgeführte Programm das Unicode-System unterstützt.

Unicode strebt die möglichst vollständige Erfassung aller bekannten Zeichen aus gegenwärtigen und vergangenen Schriftkulturen an. Die Zeichen werden nach Klassen katalogisiert und erhalten einen Zeichenwert. Alle nur erdenklichen Zeichen und Zeichensorten werden erfasst. Für Steuerzeichen wie Silbentrennzeichen, erzwungene Leerzeichen oder Tabulatorzeichen gibt es Unicodes. Die Zeichen mathematischer Formeln fehlen ebenso wenig wie die Silben- oder Wortzeichen fernöstlicher Schriftkulturen. Auch Einzelteile von Zeichen, wie etwa die Doppelpunkte über den deutschen Umlauten, haben einen eigenen Unicode. Zeichen lassen sich auch dynamisch kombinieren – so gibt es zwar natürlich auch ein deutsches »ä«, aber der gleiche Buchstabe lässt sich auch aus »a« und dem Element für Doppelpunkt über dem Zeichen erzeugen.

Neben der bloßen Adressierung eines Zeichens oder Elements ist im Unicode-System für jedes Zeichen auch ein Set von Eigenschaften definiert. Zur Eigenschaft eines Zeichens gehört z.B. die Schreibrichtung (bei arabischen Zeichen etwa ist die Schreibrichtung von rechts nach links). Insgesamt stecken hinter dem Unicode-System unzählige Forschungsergebnisse der weltweiten Sprachwissenschaft.

Das Unicode-Konsortium, das 1991 gegründet wurde und aus Linguisten und anderen Fachleuten besteht, ermittelt die aufzunehmenden Zeichen. Die vergebenen Zeichenwerte haben verbindlichen Charakter. Seit Version 2.0 ist das Unicode-System auch mit der internationalen Norm ISO/IEC 10646 synchronisiert. Das ist insofern wichtig, als HTML seit Version 4.0 und auch XML ab Version 1.0 auf der Norm ISO/IEC 10646 aufsetzen. Wenn Sie also wissen wollen, wie man ein bestimmtes Zeichen in HTML oder XML notieren soll, müssen Sie in den Unicode-Zeichentabellen nachsehen, welchen Zeichenwert das gewünschte Zeichen hat. Anschließend können Sie das gewünschte Zeichen durch eine numerische Notation wie z.B. `&#9871;` (dezimale Schreibweise) oder `&#x268F;` (hexadezimale Schreibweise mit x) im Quelltext der HTML- oder XML-Datei notieren.

Die Zeichenwerte der von Unicode erfassten Zeichen wurden bis vor kurzem noch ausschließlich durch eine zwei Byte lange Zahl ausgedrückt. Auf diese Weise lassen sich bis zu 65536 verschiedene Zeichen in dem System unterbringen (2 Byte = 16 Bit =  $2^{16}$  Kombinationsmöglichkeiten). In der Unicode-Vollversion 3.0 vom September 1999 wurden bereits 49.194 Zeichen aus aller Welt aufgelistet – die Grenze von 65536 war also schon zum Greifen nah. In der Version 3.1 vom März 2001 wurden dann nochmals 44.946 Zeichen neu aufgenommen, z.B. Zeichen aus historischen Schriften. Und so kennt die Version 3.1 also bereits 94.140 Zeichen, weshalb die Zwei-Byte-Grenze durchbrochen werden musste. Das Zwei-Byte-Schema, im Unicode-System als Basic Multilingual Plane (BMP) bezeichnet, wird deshalb von einem Vier-Byte-Schema abgelöst, wodurch sich die beruhigende Anzahl von 4.294.967.296 Zeichen ( $2^{32}$  Kombinationsmöglichkeiten für Bits) adressieren lässt.

Informationen zum Unicode-Konsortium finden Sie im Web auf der Homepage des Unicode-Konsortiums ([www.unicode.org/](http://www.unicode.org/)).

Das große Problem mit Unicode ist eigentlich nur, wie all die vielen Zeichen an einem Computer dargestellt werden sollen. Denn Unicode definiert nur Zeichenwerte und Eigenschaften von Zeichen, aber es enthält ebenso wenig wie herkömmliche Zeichensätze Angaben darüber, wie das Zeichen darzustellen ist. Dazu sind am Computer Schnittstellen wie Schriftarten erforderlich. Die klassischen Computerschriftarten sind dazu jedoch nicht geeignet, da sie sich weitgehend an bestimmten Zeichensätzen orientieren. Neue, Unicode-orientierte Schriftarten verbreiten sich allmählich. In Verbindung mit modernen Betriebssystemen und Anwendungen unterstützen solche Schriftarten zumindest die zwei-byte-breite Adressierung, also den BMP-Anteil des Unicode-Systems.

### 23.3.2 Aufbau des Unicode-Systems

Bei neuen Unicode-Versionen wird das Buch *The Unicode Standard*, herausgegeben vom Unicode-Konsortium, neu aufgelegt. Im internationalen Buchhandel ist dieses Buch erhältlich. Darin sind alle Zeichen, Zeichenwerte, Zeichenklassen usw. genau aufgeschlüsselt und dargestellt. Dieses Buch ist das verbindliche Normwerk. Auf den Web-Seiten des Unicode-Konsortiums finden sich zum schnellen Nachschlagen aber immerhin PDF-Dateien mit den einzelnen Zeichenwert-Tabellen. Die Verweise in der Tabelle weiter unten führen direkt zu den entsprechenden PDF-Dateien. Um diese Dateien betrachten zu können, benötigen Sie den Acrobat Reader von Adobe (der Web-Browser allein reicht nicht).

Das Unicode-System ist in Zahlenbereiche aufgeteilt. Die Zahlen selbst werden in der Form U+XXXX notiert. Das U steht für Unicode, und das X für je eine hexadezimale Ziffer. Zeichenwerte sind in diesen Tabellen also hexadezimal dargestellt. Falls Sie für die Notation eines oder mehrerer Zeichen aus dem Unicode-Bereich in HTML oder XML eine dezimale Darstellung verwenden wollen, können Sie den Dezimal-Hexadezimal-Umrechner auf der Buch-CD benutzen (in SELFHTML).

Die einzelnen Zeichen im Unicode-System sind nicht wahllos angeordnet. Das gesamte System ist in Zeichenbereiche aufgeteilt. Die Zeichenbereiche spiegeln jeweils eine bestimmte Schriftkultur oder ein Set von Sonderzeichen wider.

Im Folgenden werden die einzelnen Unicode-Zeichenbereiche aufgelistet. Bei jedem Zeichenbereich finden Sie einen Verweis. Der Verweis führt jeweils zu einer PDF-Datei auf der Homepage des Unicode-Konsortiums. Darauf finden Sie zu dem betreffenden Zeichenbereich Hinweise sowie eine grafische Tabelle mit Abbildungen der einzelnen Zeichen des Bereichs und der zugehörigen Unicode-Zeichenwerte (hexadezimal).

Eine grafische Darstellung zu den folgenden Teilen des Unicode-Systems in Form von PDF-Dateien finden Sie unter der Adresse [www.unicode.org/charts/](http://www.unicode.org/charts/).

| Zeichensatz                     | Zeichenbereich (hexadezimal) |
|---------------------------------|------------------------------|
| Basic Latin (ASCII-Zeichensatz) | U+0000 to U+007F             |
| Latin-1                         | U+0080 to U+00FF             |
| Latin Extended-A                | U+0100 to U+017F             |
| Latin Extended-B                | U+0180 to U+024F             |
| IPA Extensions                  | U+0250 to U+02AF             |

| <i>Zeichensatz</i>                                                                                                                                  | <i>Zeichenbereich (hexadezimal)</i> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|
| Spacing Modifier Letters                                                                                                                            | U+02B0 to U+02FF                    |
| Combining Diacritical Marks                                                                                                                         | U+0300 to U+036F                    |
| Greek                                                                                                                                               | U+0370 to U+03FF                    |
| Cyrillic                                                                                                                                            | U+0400 to U+04FF                    |
| Armenian                                                                                                                                            | U+0530 to U+058F                    |
| Hebrew                                                                                                                                              | U+0590 to U+05FF                    |
| Arabic                                                                                                                                              | U+0600 to U+06FF                    |
| Devanagari                                                                                                                                          | U+0900 to U+097F                    |
| Bengali                                                                                                                                             | U+0980 to U+09FF                    |
| Gurmukhi                                                                                                                                            | U+0A00 to U+0A7F                    |
| Gujarati                                                                                                                                            | U+0A80 to U+0AFF                    |
| Oriya                                                                                                                                               | U+0B00 to U+0B7F                    |
| Tamil                                                                                                                                               | U+0B80 to U+0BFF                    |
| Telugu                                                                                                                                              | U+0C00 to U+0C7F                    |
| Kannada                                                                                                                                             | U+0C80 to U+0CFF                    |
| Malayalam                                                                                                                                           | U+0D00 to U+0D7F                    |
| Thai                                                                                                                                                | U+0E00 to U+0E7F                    |
| Lao                                                                                                                                                 | U+0E80 to U+0EFF                    |
| Tibetan                                                                                                                                             | U+0F00 to U+0FBF                    |
| Georgian                                                                                                                                            | U+10A0 to U+10FF                    |
| Hangul Jamo                                                                                                                                         | U+1100 to U+11FF                    |
| Latin Extended Additional                                                                                                                           | U+1E00 to U+1EFF                    |
| Greek Extended                                                                                                                                      | U+1F00 to U+1FFF                    |
| General Punctuation                                                                                                                                 | U+2000 to U+206F                    |
| Superscripts and Subscripts                                                                                                                         | U+2070 to U+209F                    |
| Currency Symbols (Hier ist auch das Euro-Zeichen-Symbol mit dabei – es hat den Wert U+20AC. In HTML dezimal otiert: &#8364; oder einfacher: &euro;) | U+20A0 to U+20CF                    |
| Combining Diacritical Marks for Symbols                                                                                                             | U+20D0 to U+20FF                    |
| Letterlike Symbols                                                                                                                                  | U+2100 to U+214F                    |
| Number Forms                                                                                                                                        | U+2150 to U+218F                    |
| Arrows                                                                                                                                              | U+2190 to U+21FF                    |
| Mathematical Operators                                                                                                                              | U+2200 to U+22FF                    |
| Miscellaneous Technical                                                                                                                             | U+2300 to U+23FF                    |
| Control Pictures                                                                                                                                    | U+2400 to U+243F                    |
| Optical Character Recognition                                                                                                                       | U+2440 to U+245F                    |
| Enclosed Alphanumerics                                                                                                                              | U+2460 to U+24FF                    |
| Box Drawing                                                                                                                                         | U+2500 to U+257F                    |
| Block Elements                                                                                                                                      | U+2580 to U+259F                    |
| Geometric Shapes                                                                                                                                    | U+25A0 to U+25FF                    |
| Miscellaneous Symbols                                                                                                                               | U+2600 to U+26FF                    |
| Dingbats                                                                                                                                            | U+2700 to U+27BF                    |
| CJK Symbols and Punctuation                                                                                                                         | U+3000 to U+303F                    |
| Hiragana                                                                                                                                            | U+3040 to U+309F                    |



| <i>Zeichensatz</i>                                                                  | <i>Zeichenbereich (hexadezimal)</i> |
|-------------------------------------------------------------------------------------|-------------------------------------|
| Katakana                                                                            | U+30A0 to U+30FF                    |
| Bopomofo                                                                            | U+3100 to U+312F                    |
| Hangul Compatibility Jamo                                                           | U+3130 to U+318F                    |
| Kanbun                                                                              | U+3190 to U+319F                    |
| Enclosed CJK Letters and Months                                                     | U+3200 to U+32FF                    |
| CJK Compatibility                                                                   | U+3300 to U+33FF                    |
| CJK Unified Ideographs                                                              | U+4E00 to U+9FA5                    |
| Hangul Syllables                                                                    | U+AC00 to U+D7A3                    |
| High Surrogates                                                                     | U+D800 to U+DB7F                    |
| Private Use High Surrogates                                                         | U+DB80 to U+DBFF                    |
| Low Surrogates                                                                      | U+DC00 to U+DFFF                    |
| Private Use Area                                                                    | U+E000 to U+F8FF                    |
| CJK Compatibility Ideographs                                                        | U+F900 to U+FAFF                    |
| Alphabetic Presentation Forms                                                       | U+FB00 to U+FB4F                    |
| Arabic Presentation Forms-A                                                         | U+FB50 to U+FDFF                    |
| Combining Half Marks                                                                | U+FE20 to U+FE2F                    |
| CJK Compatibility Forms                                                             | U+FE30 to U+FE4F                    |
| Small Form Variants                                                                 | U+FE50 to U+FE6F                    |
| Arabic Presentation Forms-B                                                         | U+FE70 to U+FEFF                    |
| Halfwidth and Fullwidth Forms                                                       | U+FF00 to U+FFFF                    |
| Specials                                                                            | U+FFFO to U+FFFF                    |
| Erweiterter Bereich des Unicode-Systems (mit vier Byte breiten Zeichen darstellbar) |                                     |
| Old Italic                                                                          | U+10300 to U+1032F                  |
| Gothic                                                                              | U+10330 to U+1034F                  |
| Deseret                                                                             | U+10400 to U+1044F                  |
| Byzantine Musical Symbols                                                           | U+1D000 to U+1D0FF                  |
| Mathematical Alphanumeric Symbols                                                   | U+1D400 to U+1D7FF                  |
| CJK Unified Ideographs Extension B                                                  | U+20000 to U+2A6D6                  |

## 23.4 Downloadbare Schriftarten

### 23.4.1 Allgemeines zu downloadbaren Schriftarten

Downloadbare Schriftarten werden in Web-Seiten als Referenz auf eine Schriftartendatei mit eingebunden. Web-Browser, die downloadbare Schriftarten interpretieren, erkennen solche Schriftarten und laden sie zusammen mit den HTML- und sonstigen Ressourcendateien einer Web-Seite herunter. Nach derzeitigem Stand der Dinge wird die Schriftart dabei jedoch nicht auf dem Anwenderrechner installiert, sondern ist nur verfügbar, wenn die betreffende Web-Seite vom Originalserver angezeigt wird. Der Browser kann die Schriftartendaten interpretieren und stellt Texte, die in HTML/CSS mit dieser Schriftart formatiert sind, entsprechend dar. Voraussetzung dafür ist natürlich eine grafische Benutzeroberfläche.

Leider gibt es im Fall der downloadbaren Schriftarten keinen Standard, sondern zwei inkompatible, letztlich proprietäre Lösungen: die eine stammt von Bitstream (diese Lösung wird vom Netscape-Browser unterstützt), und die andere von Microsoft (vom Internet Explorer unterstützt). Für beide Lösungen gibt es je eine spezielle Software. Beide Softwareprogramme erlauben es unter anderem, vorhandene herkömmliche Schriftarten wie TrueType- oder Adobe-Schriftarten in das neue Dateiformat zu konvertieren oder einzelne, benötigte Zeichen in die zu erstellende Download-Schriftart zu importieren. Um für beide Browser zu schreiben, gibt es zwei Lösungen: Entweder Sie erstellen beide Typen von downloadbaren Schriftarten, arbeiten also mit beiden Programmen parallel, oder Sie beschränken sich auf die Bitstream/Truedoc-Technik, denn für den MS Internet Explorer gibt es ein ActiveX-Control, das diesen Browser befähigt, ebenfalls die Bitstream/Truedoc-Technik zu nutzen.

Im Bereich des Web-Designs haben sich die downloadbaren Schriftarten nach mehreren Jahren ihres Existierens nicht sonderlich durchgesetzt. Der Grund ist wohl einerseits das etwas ungewohnte Handling mit dem Kopierschutz bei solchen Schriftarten (die Schriftarten werden nur interpretiert, wenn sie von einer frei wählbaren, aber bestimmten Domain-Adresse geladen werden). Andererseits hat man aber auch eingesehen, dass die meisten kunstvollen Schriftarten auf pixelorientierten Bildschirm sowieso nicht so gut aussehen wie auf Papier. Während also nur wenige Web-Seiten aus Design-Gründen auf downloadbare Schriftarten setzen, können diese dagegen eine durchaus sinnvolle Lösung für das Problem der Internationalisierung sein. Web-Seiten, die in z.B. kyrillische Zeichen enthalten, könnten downloadbare Schriftarten benutzen, um sicherzustellen, dass die Seiten auch auf westeuropäischen und amerikanischen Computern korrekt angezeigt werden. Freilich ist auch für diese Fälle durch das Unicode-System und modernere Schriftarten, die große Teile des Unicode-Systems abdecken, eine gewisse Verbesserung eingetreten.

### 23.4.2 TrueDoc (Bitstream)

Bitstream ist ein Unternehmen, das Lösungen für die Schriftdarstellung in elektronischen Dokumenten entwickelt ([www.bitstream.com/](http://www.bitstream.com/)). Zu den Technologien, die dort entwickelt werden, gehört auch diejenige mit dem Namen TrueDoc. Auf den Web-Seiten zu TrueDoc finden Sie alle weiteren nötigen Originalinformationen zur TrueDoc-Technologie.

Hinter TrueDoc verbirgt sich ein Schriftartenkonzept für plattformübergreifendes elektronisches Publizieren speziell auch für den Einsatz im World Wide Web. TrueDoc-Schriftarten werden nicht wie herkömmliche Schriftarten beim Anwender installiert und sind dann dort »verfügbar«, sondern sie werden direkt in elektronische Dokumente mit eingebunden oder darin referenziert. Software, die TrueDoc interpretiert, erkennt solche Schriftarten und lädt sie zusammen mit den anzuzeigenden Daten eines Dokuments. Dokumentdaten, die mit TrueDoc-Schriftarten formatiert sind, werden dann am Anwenderbildschirm entsprechend dargestellt.

Beim Einsatz im Web sind zum Funktionieren der TrueDoc-Technologie folgende Voraussetzungen erforderlich:

- Als Web-Autor benötigen Sie die kostenpflichtige Entwicklersoftware für TrueDoc-Schriftarten von Bitstream. Über die Web-Seiten von TrueDoc können Sie diese Software erwerben.
- Die Schriftarten nach dem TrueDoc-Modell müssen nach einer bestimmten Konvention in Web-Seiten eingebunden werden. Der Einsatz ist serverabhängig. Das heißt, beim Erstellen

der Schriftarten bestimmen Sie, von welchen URIs aus die Schriftarten heruntergeladen werden dürfen. Funktionieren tun sie dann nur, wenn die Web-Seite über entsprechende Adressen aufgerufen wird. Auf diese Weise wird Missbrauch und Raubkopieren der Schriftarten unterbunden. Das entsprechende Sicherheitskonzept nennt sich bei TrueDoc DocLock. Wie das DocLock-Konzept im Einzelnen funktioniert, gehört zur Softwaredokumentation der TrueDoc-Entwicklersoftware.

- Der Web-Browser des Anwenders muss die TrueDoc-Technologie unterstützen. Netscape hat die TrueDoc-Technologie seit Version 4.01 seines Web-Browsers adoptiert. Microsoft kontert mit der Konkurrenz-Software *WEFT*. Andererseits ermöglicht Bitstream den Entwicklern von TrueDoc-Schriften, ein nicht sehr großes ActiveX-Control mit anzubieten, das den MS Internet Explorer 4.0 oder höher ebenfalls befähigt, Web-Seiten mit eingebundenen TrueDoc-Schriftarten zu interpretieren.

Schriftartendateien der TrueDoc-Technologie haben die Dateiendung *.pfr* (*»Portable Font Resource«*).

### 23.4.3 Microsoft und das Web Embedding Fonts Tool WEFT

Als Technologie für downloadbare Schriftarten hat Microsoft das Programm WEFT (*Web Embedding Fonts Tool*) entwickelt. Der MS Internet Explorer unterstützt ab der Version 4.0 seines Browsers diese Technologie. Sie können das Programm WEFT auf der Microsoft WEFT Homepage downloaden ([www.microsoft.com/typography/web/embedding/weft/](http://www.microsoft.com/typography/web/embedding/weft/)). An der gleichen Stelle finden Sie auch eine Einführung in die WEFT-Technologie.

Um WEFT sinnvoll nutzen zu können, erstellen Sie zuerst Ihr Web-Projekt und verwenden innerhalb Ihrer HTML-Dateien mit den dafür üblichen Möglichkeiten in HTML- bzw. CSS gewünschte Schriftartennamen. Anschließend können Sie den WEFT Wizard starten (Teil von WEFT). Mit diesem Software-Assistenten können Sie das Projekt analysieren lassen. Der WEFT Wizard ermittelt die verwendeten Schriftartennamen. Sie können auswählen, für welche der Schriftarten Sie ein *»Font Object«*, sprich eine Schriftartendatei, erstellen möchten.

Schriftartendateien, die mit WEFT erzeugt werden, haben die Dateiendung *.eot*. Beim Erzeugen dieser Schriftarten gibt es diverse Optionen, etwa die, ob Sie für jede HTML-Datei eine eigene Schriftart oder eine gemeinsame Schriftart für alle HTML-Dateien wünschen. WEFT greift auch auf Ihre erstellten HTML-Dateien zu und ändert den Quelltext an Stellen, an denen es für die korrekte Anzeige der erstellten Schriftart erforderlich ist.

Beim Erzeugen der eot-Dateien müssen Sie außerdem deren geplanten Ablageort im Web angeben. Das kann für Web-Projekte beispielsweise eine FTP-Adresse sein, auf der Sie Dateien ablegen können. Ferner geben Sie an, welche URIs diese Schriftartendateien benutzen dürfen. Dies ist eine Maßnahme, die verhindern soll, dass jeder, der will, Ihre Schriftartendateien per Referenz in seinen eigenen Projekten verwendet. Die WEFT-Technologie funktioniert so weit, ist jedoch auf den Internet Explorer beschränkt – d.h. Anwender mit anderen Browsern haben nichts davon.

### 23.4.4 Downloadbare Schriftarten in HTML einbinden

Um innerhalb einer HTML-Datei downloadbare Schriftarten zu verwenden, müssen Sie die Schriftarten nach einem der folgenden Beispiele referenzieren.

Beispiel 1 – *pfr*-Dateien (TrueDoc) nach HTML-Syntax einbinden:

```
<html><head><title>Test</title>
<link rel="fontdef" src="http://www.meine.com/fonts/chianti.pfr">
</head><body>
<font face="Chianti">Text in Beispielschriftart</font>
</body></html>
```

Beispiel 2 – *pfr*-Dateien (TrueDoc) nach CSS-Syntax einbinden:

```
<html><head><title>Test</title>
<style type="text/css">
@font-face { font-family:Chianti XBd BT;
              src:url(http://www.meine.com/fonts/chianti.pfr); }
</style>
</head><body>
<p style="font-family:Chianti XBd BT">Text in Beispielschriftart</p>
</body></html>
```

Beispiel 3 – *eot*-Dateien (Microsoft WEFT) nach CSS-Syntax einbinden:

```
<html><head><title>Test</title>
<style type="text/css">
@font-face { font-family:Chianti XBd BT;
              src:url(http://www.meine.com/fonts/chianti.eot); }
</style>
</head><body>
<p style="font-family:Chianti XBd BT">Text in Beispielschriftart</p>
</body></html>
```

Downloadbare Schriftarten werden im Kopf einer HTML-Datei eingebunden, die solche Schriftarten verwendet. Eine downloadbare Schriftart nach Bitstream-Technologie (PFR-Datei) können Sie entweder mit einem `<link>`-Tag (wie in Beispiel 1) oder mit einer CSS Stylesheet-Definition (wie in Beispiel 2) einbinden. Benutzen Sie das `<link>`-Tag, wenn Sie in HTML mit `<font face=>` arbeiten. Davon ist aus heutiger Sicht allerdings abzuraten. Besser ist es, wenn Sie mit CSS arbeiten wie in Beispiel 2 gezeigt. Wenn Sie beide Formen benutzen, binden Sie die Schriftart nach beiden Mustern ein.

Das Einbinden von *eot*-Dateien nach der Microsoft-WEFT-Technologie brauchen Sie nicht selbst durchzuführen. Einträge von der Art, wie sie im obigen Beispiel 3 notiert sind, nimmt das Programm WEFT selbstständig in den betroffenen HTML-Dateien vor. Es ist dann egal, ob Sie die Schriftart innerhalb der HTML-Datei als CSS Stylesheet-Angabe oder traditionell mit `<font face=>` verwenden. Der MS Internet Explorer 4 erkennt in jedem Fall den Bezug und überträgt die erforderliche Schriftart mit auf den Rechner des Anwenders.

**Beachten Sie:** Allgemeine Beschreibungen zu den oben gezeigten HTML- und CSS-Notationen finden Sie innerhalb dieser Dokumentation an folgenden Stellen:

- Schriftarten mit Schriftartendatei in CSS Stylesheets (CSS-Syntax zum Einbinden von Schriftartdateien, Kapitel 16.2).
- Logische Beziehungen einer HTML-Datei (HTML-Syntax für logische Verknüpfung zu anderen Dateien, Kapitel 4.1.12).

Die Bitstream TrueDoc-Schriftarten werden von Web-Servern nur dann korrekt an den aufrufenden Browser übertragen, wenn in der Konfiguration des Web-Servers ein Mime-Type für den entsprechenden Dateityp eingetragen ist. Bitten Sie gegebenenfalls Ihren Webmaster/Provider/Administrator, den erforderlichen Eintrag `application/font-tdpfr` für Dateien mit der Endung `.pfr` in der Konfiguration des Web-Servers für Sie vorzunehmen, oder tragen Sie diesen Mime-Type selber ein, sofern Sie Zugriff auf die Web-Server-Konfiguration haben.

## 24 Grafik

### 24.1 Grafikformate für Web-Seiten

#### 24.1.1 Allgemeines zu Grafikformaten für Web-Seiten

Im Web haben sich zwei Dateiformate für Grafiken auf breiter Front durchgesetzt, die von allen, auch den nicht so modernen Browsern »inline«, also im eigenen Fenster angezeigt werden können: die Formate GIF und JPEG. Die Stärken beider Formate liegen jedoch auf unterschiedlichen Gebieten. Dazu kommt ein neues, viel versprechendes Format, das viele Vorteile von GIF und JPEG in sich vereint: das PNG-Format. Dieses Format wird von moderneren Browsern ebenfalls unterstützt, wenn auch noch uneinheitlich und oft nicht vollständig. Daneben existieren noch andere, beachtenswerte Lösungen, die aber nur einen kleinen Benutzerkreis haben, weil sie bei Browsern ein Plug-In benötigen – z.B. das Format von LuraWave. Eines aber haben all diese Formate gemeinsam: Es sind pixelorientierte Formate. Vektorgrafik ist im Web dagegen noch eine Seltenheit. Das dafür viel versprechendste Format, das auf XML basierende SVG-Format, entwickelt sich leider nur zögerlich. Es bleibt die Hoffnung, dass es dennoch angenommen wird. In diesem Abschnitt werden die genannten Grafikformate kurz vorgestellt.

#### 24.1.2 Das GIF-Format

Dateien des GIF-Formats haben die übliche Dateinamen-Endung *.gif* und den Mime-Type *image/gif*. In HTML lassen sich GIF-Grafiken mit Hilfe des *img*-Elements einbinden.

GIF steht für Graphics Interchange Format und wurde schon vor vielen Jahren vom Online-Anbieter CompuServe eingeführt. Es zeichnet sich durch eine hohe Kompression aus. Deshalb hat es sich im Online-Bereich, wo die Übertragung von Daten Geld und Zeit kostet, schnell durchgesetzt. Der heute weit verbreitete Standard des GIF-Formats ist das so genannte »89er-Format«. Dieses Format bietet drei Möglichkeiten an, die das GIF-Format für den Einsatz im Web besonders interessant machen:

- **Interlaced:** Eine Grafik, die mit der Option »interlaced« abgespeichert ist, wird beim Laden nicht zeilenweise eingelesen und aufgebaut, sondern schichtweise. Gerade beim Laden von Web-Seiten aus dem Netz ist das sehr hilfreich. Denn so erscheint die Grundstruktur der Grafik sehr schnell am Bildschirm des Anwenders. Beim weiteren Ladevorgang wird die Grafik dann immer deutlicher und feiner aufgelöst am Bildschirm angezeigt.
- **Animation:** Das GIF-Format bietet die Möglichkeit, mehrere Grafiken in einer einzigen Grafikdatei zu speichern, verbunden mit Optionen zur zeitlichen Steuerung der Einzelgrafiken. Dadurch werden animierte GIF-Grafiken möglich, siehe Kapitel 24.4.
- **Transparente Farbe:** Das GIF-Format arbeitet mit Farbpaletten, also mit einer begrenzten Anzahl »indizierter Farben«. Dabei besteht die Möglichkeit, eine dieser Farben als »transpa-

rent« zu definieren. Bei der Anzeige scheinen dann Elemente, die unterhalb der Grafik liegen, an den Stellen durch, wo die Grafik transparent ist. Einzelheiten zu den damit verbundenen Möglichkeiten werden im Kapitel 24.3.6 *Transparenz einer bestimmten Palettenfarbe* beschrieben.

Ein Nachteil des GIF-Formats ist, dass es maximal 256 Farben pro Datei speichern kann. Ein Pluspunkt ist dagegen, dass GIF-Grafiken verlustfrei komprimieren. Aufgrund dieser Charakteristika ist das GIF-Format für hochauflösende Grafiken wie Fotos nicht so sehr geeignet. Ideal ist das GIF-Format dagegen für typische Web-Grafiksorten wie Buttons, Dots, Bars, Symbole und Cliparts.

Leider ist auch das GIF-Format nicht mehr ganz frei vom Schmutz der Abzockversuche im Internet. Die Firma Unisys, die ursprünglich den LZW-Algorithmus patentierte, den das GIF-Format benutzt, wirbelte Ende 1999 viel Staub auf, als sie bekannt gab, juristisch gegen nicht lizenzierte GIF-Grafiken im Web vorzugehen. Angesichts der Tatsache, dass es wohl in etwa so viele GIF-Grafiken im Web gibt wie Menschen auf der Erde, war dieses Vorhaben allerdings nicht besonders realistisch. Anbieter von Grafiksoftware, die das GIF-Format unterstützt, sind allerdings lizenzpflchtig und werden auch kontrolliert. Deshalb gibt es vor allem in Freeware-Kreisen noch mancherorts Proteste und Aufrufe, das GIF-Format nicht mehr zu verwenden.

### 24.1.3 Das JPEG-Format

Dateien des JPEG-Formats haben die übliche Dateinamen-Endung *.jpg*, *.jpeg* oder *.jpe* und den MIMEType `image/jpeg`. In HTML lassen sich JPEG-Grafiken mit Hilfe des `img`-Elements einbinden.

JPEG steht für Joint Photographic Expert Group, also nicht der Name des Grafikformats, sondern der Name der Firma, die das Format entwickelt hat. Das JPEG-Verfahren ist ein Kompressionsalgorithmus für Datenströme, der auf dem Algorithmus DCT (Diskrete Cosinus Transformation) in Verbindung mit der Huffman-Kodierung basiert. Das gleichnamige Dateiformat für Grafiken ist einfach eine Anwendung dieses Algorithmus auf Pixelgrafiken. Mittlerweile wird der JPEG-Algorithmus auch auf Videos angewendet und hat das beliebte Videoformat MPEG hervorgebracht.

Das JPEG-Grafikformat komprimiert wie das GIF-Format ebenfalls sehr gut und hat gegenüber dem GIF-Format den Vorteil, dass es pro Bild bis zu 16,7 Millionen Farben speichern kann. Es arbeitet deshalb nicht wie das GIF-Format mit Farbpaletten bestimmter Farben, sondern mit dem gesamten Farbspektrum. Der Nachteil bei JPEG ist, dass es mit Verlust komprimiert. Je höher der Kompressionsfaktor, desto schlechter die Qualität der Grafik.

Beim JPEG-Format können Sie bei besseren Grafikprogrammen zwei Parameter selbst bestimmen:

- **DPI-Dichte:** Das ist die Punktdichte (dpi = dots per inch). Bei JPEG-Dateien, die Sie nur für die Anzeige in Web-Seiten benötigen, genügt ein Wert zwischen 70 und 100 dpi. Falls Sie die Grafik ausdrucken wollen, benötigen Sie allerdings deutlich mehr dpi.
- **Kompressionsfaktor:** Je höher der Kompressionsfaktor, desto schlechter die Bildqualität. Je geringer der Kompressionsfaktor, desto größer die Datei.

Verluste durch Kompression entstehen bei JPEG-Grafiken vor allem bei scharfen Farbübergängen im Bild, also bei Ecken und Kanten von abgebildeten Gegenständen (»Ränder fransen aus«). Für Bilder mit wenigen Farben, aber klaren Konturen, also z.B. für einfache Zeichnungen, scharfe Schriftzüge usw. ist das JPEG-Format deshalb nicht geeignet. Seine Stärke zeigt das Format dagegen beim Abspeichern von Fotos und anderen Grafiken, in denen sehr feine Farbverläufe vorkommen. Selbst bei geringer Kompression sind die Dateigrößen schon um ein Vielfaches kleiner als etwa bei Bitmap-Grafiken. Sogar den Bildschirm füllende Fotos lassen sich mit JPEG bei brauchbarer Qualität auf Dateigrößen bringen, die im Web als akzeptabel gelten können.

Es gibt eine Variante von JPEG-Grafiken, so genannte »progressive JPEG-Grafiken«. Dabei wird, ähnlich wie beim GIF-Format in der Variante »interlaced«, zuerst eine vollständige, aber noch undeutliche Fassung der Grafik am Bildschirm aufgebaut. Das erste, noch nicht ganz deutliche Bild kann bereits am Bildschirm angezeigt werden, nachdem etwa 15% der Grafikdatei an den aufrufenden Web-Browser übertragen wurden.

#### 24.1.4 Das PNG-Format

Dateien des PNG-Formats haben die übliche Dateinamen-Endung *.png* und den MIMEType *image/png*. In HTML lassen sich PNG-Grafiken mit Hilfe des *img*-Elements einbinden, wahlweise aber auch mit den Elementen *embed* oder *object*.

PNG bedeutet Portable Network Graphic (ausgesprochen: PING). Es handelt sich um ein Grafikformat, das eigens für den Einsatz im Web konzipiert wurde und dessen Normierung wie diejenige von HTML, CSS, XML und andere Web-Technologien vom W3-Konsortium besorgt wird. Die Empfehlung (Recommendation) des W3-Konsortiums zu PNG stammt aus dem Jahr 1996. Das entsprechende Dokument ist unter dem Titel *PNG (Portable Network Graphics) Specification* auf den Seiten des W3-Konsortiums zu finden ([www.w3.org/TR/REC-png](http://www.w3.org/TR/REC-png)).

PNG soll die Vorteile von GIF und JPEG in sich vereinen und zeichnet sich durch folgende Charakteristika aus:

- **Verlustfreie Kompression:** PNG komprimiert verlustfrei wie das GIF-Format. Der eingesetzte Kompressionsalgorithmus arbeitet dabei ähnlich wie der des ZIP-Formats. Wiederholungen von Pixelmustern im Bild werden durch Meta-Informationen über diese Muster ersetzt, was vor allem bei Grafiken wie Cliparts, Schriftzügen usw. zu hohen Kompressionsdichten führt. Weniger stark greift dieser Algorithmus dagegen bei Fotos und Grafiken mit vielen feinen Farbübergängen. Das ist ganz ähnlich wie beim Komprimieren von Daten zu ZIP-Dateien, wo sich Binärdateien mit wenigen erkennbaren Byte-Mustern weniger stark komprimieren lassen als z.B. Textdateien.
- **Farbtiefe plus Transparenz:** PNG unterstützt wie das JPEG-Format 16,7 Mio. Farben, wahlweise aber auch das Abspeichern mit indizierten Farben, also mit einer begrenzten Farbpalette. So bleibt das Format flexibel und ist für alle Arten von Grafiken nutzbar. Neben den drei Farbwerten für Rot, Grün und Blau kann das PNG-Format zusätzlich noch einen vierten Wert pro Pixel speichern, nämlich den Alpha-Kanal. Der Alphawert speichert den Transparenzgrad des Pixels. Auf diese Weise werden deutlich feiner abgestufte Transparenz-Effekte möglich, als es mit dem GIF-Format möglich ist.



- **Interlaced-Funktion:** Wie das GIF-Format und auch das JPEG-Format in der Variante »progressive« unterstützt PNG das schichtweise Aufbau des Bildes (Interlaced-Funktion). Bei PNG genügen bereits 1-2% übertragene Daten, um das Bild schemenhaft anzeigen zu können.
- **Informationen zur Gammakorrektur:** Das PNG-Format erlaubt das Abspeichern einer Reihe von Zusatzinformationen zum Bild, darunter den Gamma-Wert, der z.B. bei der Aufnahme eines Fotos maßgeblich war. Web-Browser, die das Bild anzeigen, könnten diese Information beispielsweise dazu nutzen, um bei der Anzeige des Bildes eine automatische Gammakorrektur abhängig vom eingesetzten Bildschirm und Betriebssystem vorzunehmen. Dieses Feature kann vermeiden helfen, dass ein und dieselbe Grafik auf verschiedenen Bildschirmen und unter verschiedenen Betriebssystemen oft so unterschiedlich aussieht.
- **Informationen zu Bildherkunft und Copyright:** Das PNG-Format erlaubt das Abspeichern von Textfeldern zu Bildtitel, Bildautor, Bildbeschreibung, Copyright, Erstellungsdatum, Erstellungs-Software, Datenquelle sowie zu rechtlichen und sittlichen Absicherungshinweisen.

Animierte Grafiken wie beim GIF-Format sind mit dem PNG-Format allerdings nicht möglich. Es gibt also insgesamt viele Gründe, die für das PNG-Format sprechen. Weniger als Konkurrenz zum JPEG-Format (JPEG komprimiert bei Fotos doch deutlich stärker als PNG – die PNG-Dateigrößen hoch auflösender Fotos sind nicht sonderlich webfreundlich), aber als Konkurrenz zum GIF-Format, das seit dem lizenzrechtlichen Hahn-im-Korb-Gebaren von Unisys etwas ins Zwielicht geraten ist. Das größte Problem von PNG ist eigentlich nur die zögerliche Unterstützung bei der Software. Zwar unterstützt Netscape das PNG-Format seit der Produktversion 4.04, und auch der Internet Explorer kann PNG-Grafiken seit Version 4.0 anzeigen. Auch immer mehr Grafikprogramme beherrschen das PNG-Format. Doch gab es dann bei einzelnen Zwischenversionen der Browser wieder Ausfälle bei der PNG-Unterstützung. Auch wurde das Format nicht vollständig unterstützt usw. Deshalb – und natürlich auch aus der Gewohnheit heraus – tut sich das PNG-Format noch schwer gegen das GIF-Format.

### 24.1.5 Das LuraWave-Format

Dateien des LuraWave-Formats haben die übliche Dateinamen-Endung *.lwf* und den Mime-Type `image/x-wavelet`. In HTML lassen sich LuraWave-Grafiken mit Hilfe der Elemente `embed` oder `object` einbinden.

LuraWave ist ein Dateiformat der Firma LuraTech für Pixelgrafiken. Es eignet sich vor allem für hochauflösende Grafiken, also Fotos und Bilder mit feinen Farbübergängen. Bei solchen Grafiken macht sich der besondere Kompressionsalgorithmus bemerkbar, den das LuraWave-Format verwendet, und der den Algorithmen von JPEG und PNG überlegen ist. Der Lura-Algorithmus, der mit so genannter **Wavelet-Kompression** arbeitet, bewahrt im direkten Vergleich mit JPEG bei einer gleichen Dateigrößenreduktion eines Originalbildes eine deutlich bessere Bildqualität als das JPEG-Format. Neben der besonders guten Kompression verfügt das LuraWave-Format über weitere Besonderheiten:

- **Skalierbare Größe:** Aufgrund des Wavelet-Kompressionsalgorithmus ist es möglich, ähnlich wie bei einem Hologramm das Gesamtbild aus einem kleinen Teil davon zu rekonstruieren. Das erlaubt es, eine LuraWave-Grafik zu zoomen, also zu vergrößern, ohne dass die Grafik dadurch gleich unansehnlich pixelig wird.

- **Passwortschutz:** LuraWave-Grafiken lassen sich in der Weise mit einem Passwort schützen, dass eine qualitativ minderwertigere Bildvariante öffentlich betrachtbar ist, während eine qualitativ hochwertigere Variante nur gegen Eingabe eines Passworts anzeigbar und bearbeitbar ist.

Zum Erstellen von LuraWave-Grafiken ist LuraWave-Software erforderlich. Die Software gibt es in Form von Plug-Ins für bekannte Produkte wie Photoshop, Cumulus oder Paintshop Pro. Solche Programme können dann Grafiken als lwf-Dateien abspeichern. Es gibt auch ein separates Tool von LuraWave, das andere Grafikformate wie JPEG, BMP, TIFF usw. einlesen und die Grafiken dann im lwf-Format abspeichern kann. Die Plug-Ins für die Profiprogramme und die Profiversion des separaten Grafiktools sind kostenpflichtig, eine Light-Version des separaten Tools ist kostenlos. Kostenlos sind ferner die leider erforderlichen Browser-Plug-Ins. Infos zu den Produkten und Download-Möglichkeiten gibt es auf den Web-Seiten von LuraTech ([www.luratech.com/](http://www.luratech.com/)).

Das LuraWave-Format ist also eine technisch interessante Alternative vor allem zum JPEG-Format. Nachteil ist, dass es kein offener Standard ist, von bestimmter Software abhängig ist und von Web-Browsern nicht von Haus aus, sondern nur nach der Installation des Plug-Ins unterstützt wird.

### 24.1.6 Das SVG-Format

Dateien des SVG-Formats haben die übliche Dateinamenendung `.svg` oder `.svgz` und den MIME-Type `image/svg+xml`. In HTML lassen sich SVG-Grafiken mit Hilfe der Elemente `embed` oder `object` einbinden.

SVG bedeutet Scalable Vector Graphics. Es ist das einzige vektorgrafische der hier vorgestellten Formate. Im Gegensatz zu klassischen Formaten dieser Art, die in der Regel an kommerzielle Software gebunden sind, ist das SVG-Format eine Entwicklung des W3-Konsortiums und genauso wie andere dort normierte Sprachen, also wie HTML, CSS, XML und auch PNG, plattformübergreifend, offen dokumentiert und frei verwendbar. Aktuelle Informationen und die Spezifikation von SVG finden Sie auf den Seiten des W3-Konsortiums über die Einstiegsseite Scalable Vector Graphics (SVG, [www.w3.org/Graphics/SVG/](http://www.w3.org/Graphics/SVG/)).

Vektorgrafische Formate beschreiben nicht, welche Farbanteile ein Pixel im Bild hat, sondern sie beschreiben Objekte. Es gibt Objekte wie gerade und nach einem Algorithmus gekrümmte Linien, runde und winklige Flächen, Text usw. All diese Objekte haben im Bild eine bestimmte Position und bestimmte Ausmaße, Farben und andere Eigenschaften. Flächen können einfach, mit einem Farbverlauf oder mit einem Bitmap-Muster gefüllt sein, 3-D-Effekte und Farbfilter zugewiesen haben. Objekte können auch zu Objektgruppen zusammengefasst sein. In der Datei stehen dann nur die Beschreibungen der Objekte. Die anzuzeigende Grafik wird zur Laufzeit aus den beschreibenden Daten erzeugt.

Ideal ist diese Grafikform für das Web insofern, als die Beschreibungsdaten für vektorgrafische Objekte in der Regel viel weniger Speicherplatz benötigen als pixelorientierte Grafiken. Gerade für Grafiken, die Symbole enthalten, Logos oder Cliparts, wäre das SVG-Format eine sinnvolle Alternative zum GIF-Format. Ideal geeignet ist es für jede Art von technischer Grafik, für Illustrations- und Konstruktionsgrafik. Das SVG-Format zeichnet sich durch folgende besonderen Eigenschaften aus:

- **XML-basiert:** SVG ist eine XML-basierte Sprache. Das Format ist dadurch jederzeit gegen eine DTD validierbar, und, was ganz wichtig ist – es ist ein Klartextformat. SVG-Grafiken sind daher z.B. für Suchmaschinen nach Textinhalten durchsuchbar. Ferner ist es problemlos möglich, SVG-Grafiken dynamisch mit Scripts zu erzeugen, sei es serverseitig mit CGI-Scripts, oder clientseitig mit einer Sprache wie JavaScript. Mit Hilfe des Document Object Model (DOM) ist sogar der direkte dynamische Zugriff auf jedes Beschreibungsdatum einer SVG-Datei möglich. SVG eignen sich deshalb z.B. hervorragend für interaktive Grafiken, bei denen ein Anwender in einem HTML-Formular Parameter eingibt, aus denen die Grafik dann zusammengestellt wird, oder für dynamisch erzeugte Balken- und Tortendiagramme zu Server-Statistiken.
- **XLink-Unterstützung:** XLink ist ein Schema zum Beschreiben von Verweisen (Hyperlinks) in XML. Auch auf SVG ist dieses Schema anwendbar. Dadurch ist es möglich, verweissensitive SVG-Grafiken zu erstellen. Die Verweise können zu beliebigen lokalen oder weltweiten Zielen führen.
- **Zoom- und Schwenkfähigkeit:** Vektorgrafiken lassen sich verlustfrei zoomen. So können kleine Grafiken für sehschwache Menschen problemlos vergrößert werden. Ebenso lassen sich solche Grafiken leicht drehen (schwenken).
- **SMIL-Schnittstelle:** SMIL ist ebenfalls eine vom W3-Konsortium normierte Sprache. Sie beschreibt Zeitschienen für multimediale Abläufe. Zusammen mit SMIL lassen sich dynamische SVG-Grafiken mit anderen Elementen wie Sound kombinieren.

Neuere Versionen bekannter vektorgrafischer Softwareprodukte unterstützen das SVG-Format, so etwa CorelDraw ab Version 10. Andere, als erschwingliche Sharewareprogramme vertriebene Produkte wie das Programm WebDraw des Herstellers Jasc, von dem auch das beliebte Pixelprogramm Paintshop Pro stammt, helfen ebenfalls beim Erstellen von SVG-Grafiken. Theoretisch genügt allerdings auch ein beliebiger Texteditor – so wie bei allen Klartextformaten der XML-Familie. Wer bereit ist, sich auf die SVG-Sprache einzulassen, kann tatsächlich mit Texteditor und SVG-Viewer ausgerüstet SVG-Grafiken erstellen. Und SVG-Viewer gibt es kostenlos. Der bekannteste ist von Adobe. Der Adobe-Viewer lässt sich auch als Browser-Plug-In installieren. Auf diese Weise sind Browser wie Netscape oder der Internet Explorer in der Lage, SVG-Grafiken im Web anzuzeigen. Leider fehlt jedoch noch die native Unterstützung von SVG durch die Browser. Alles, wofür der Anwender ein Plug-In braucht, entwickelt sich nur zögerlich. Eine im Browser eingebaute SVG-Unterstützung so wie für GIF und JPEG wäre daher dringend wünschenswert. Denn das SVG-Format ist vermutlich die spannendste Entwicklung im Bereich Grafik, die derzeit überhaupt stattfindet, und es eignet sich wie kaum ein anderes für etliche statische und dynamische Grafiksorten im Web.

## 24.2 Typische Grafiksorten für Web-Seiten

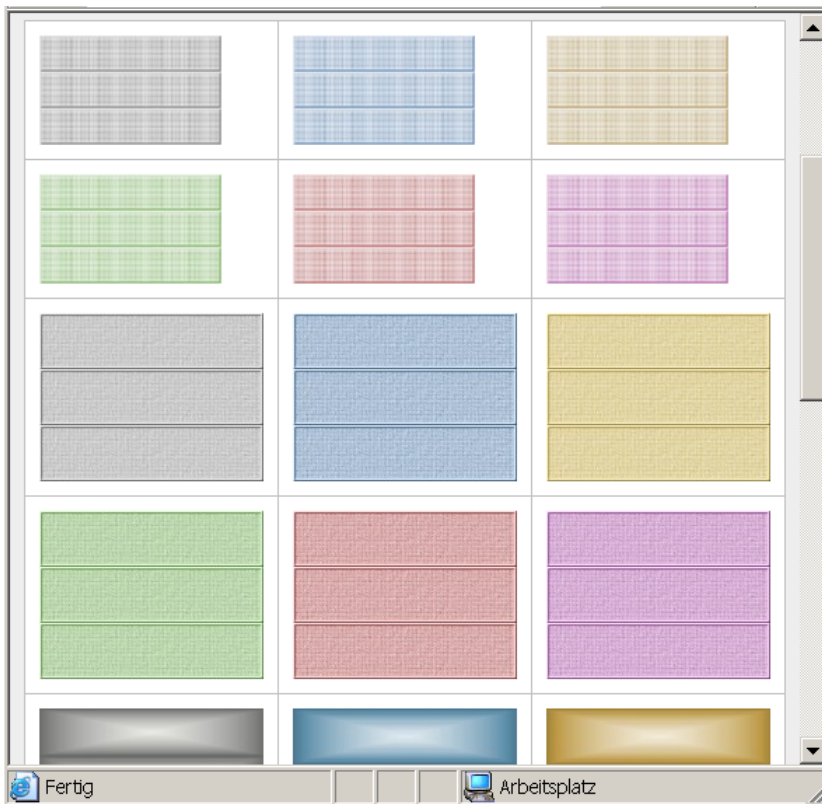
### 24.2.1 Buttons (Schaltflächen)

Buttons werden auf Web-Seiten gerne verwendet, um Verweise grafisch und attraktiv darzustellen. Achten Sie dabei jedoch immer auf die gesamte zu ladende Datenmenge. Bei einem halben Dutzend Verweisen sind kleine grafische Buttons durchaus akzeptabel, nicht aber bei Dutzenden von Verweisen auf einer Seite.

Näheres zum Einbinden in HTML wird in Kapitel 7.3 *Grafiken als Verweise* beschrieben. Verwenden Sie bei mehreren Buttons auf einer Web-Seite möglichst eine einheitliche Farbpalette! Das typische HTML-Konstrukt für anklickbare Verweis-Buttons lautet:

```
<a href="datei.html">

</a>
```

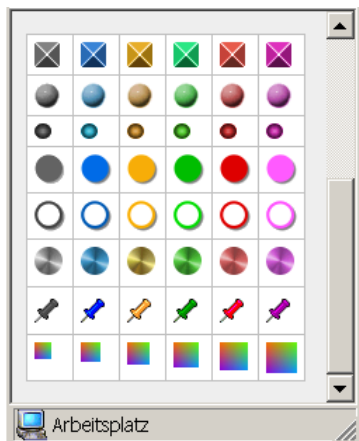


24.1: Buttons finden vor allem als Links Verwendung.

### 24.2.2 Dots (Auflistungspunkte)

Nur Text am Bildschirm ist langweilig und ermüdend. Das Auge sucht nach Augenweiden. Mit kleinen, farbigen Punkten, Quadraten, Dreiecken und informellen Flecken, die aufgrund ihrer Größe nur wenig Speicher brauchen, können Sie textlastige Information augenfreundlich unterstützen. Wichtig ist hierbei zu beachten, dass der Leser dazu neigt, den Dots irgendeine Bedeutung zuzuordnen, z.B. »wo dieser blaue Punkt steht, folgt ein Produktname«. Verwenden Sie Dots daher so, dass diese eine nachvollziehbare Bedeutung haben. Verwenden Sie nicht lauter unterschiedliche Dots auf einer Web-Seite, sondern wiederholen Sie die gleichen Dots mehrmals

pro Web-Seite. Das ist nicht langweilig, sondern sorgt für ein sauberes Erscheinungsbild und spart beim Anwender Ladezeit, da die betreffende Grafik nur einmal geladen werden muss.



24.2: Die Aufzählungspunkte sind auch als Dots oder Bullets bekannt.

Dots sind zum Beispiel in Aufzählungslisten sinnvoll und lassen sich mit Hilfe der CSS-Eigenschaft `list-style-image` sogar ganz elegant einbinden. Ein typisches HTML-Konstrukt sieht dann etwa so aus:

```
<ul style="list-style-image:url(dot.gif)">
<li>Listenpunkt</li>
<li>anderer Listenpunkt</li>
</ul>
```

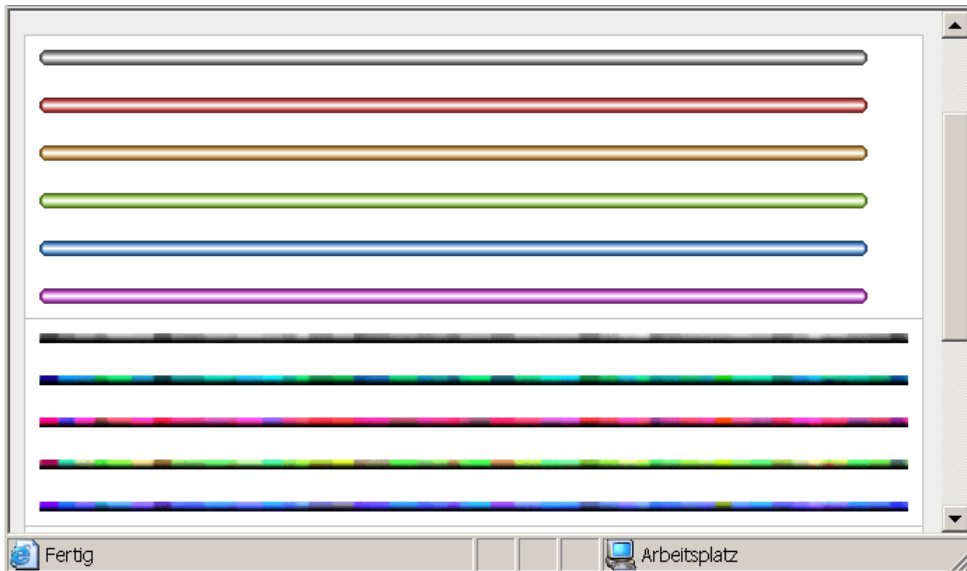
### 24.2.3 Bars (Trennleisten)

HTML bietet zwar die Möglichkeit an, Trennlinien in den Text einzufügen, um Information übersichtlicher zu strukturieren. Wer auf seinen Web-Seiten jedoch etwas Besonderes bieten will, verwendet grafische Linien. Ähnlich wie Dots benötigen Bars wenig Speicher. Auch bei Bars gilt: Verwenden Sie nicht lauter unterschiedliche Bars auf einer Web-Seite, sondern wiederholen Sie die gleichen Bars mehrmals pro Web-Seite. Das ist nicht langweilig, sondern sorgt für ein sauberes Erscheinungsbild und spart beim Anwender Ladezeit, da die betreffende Grafik nur einmal geladen werden muss.

Bei sauber kodiertem HTML sollten Grafiken innerhalb anderer Block-Elemente notiert werden. Ein typisches HTML-Konstrukt zum Notieren einer Bar, das mit den CSS-Eigenschaften `margin-top` und `margin-bottom` auch gleich Abstände zum vorhergehenden und nachfolgenden Inhalt definiert, könnte etwa so aussehen:

```
<div style="margin-top:20px; margin-bottom:20px;">

</div>
```



24.3: Trennlinien helfen bei einer optisch ansprechenden Gliederung

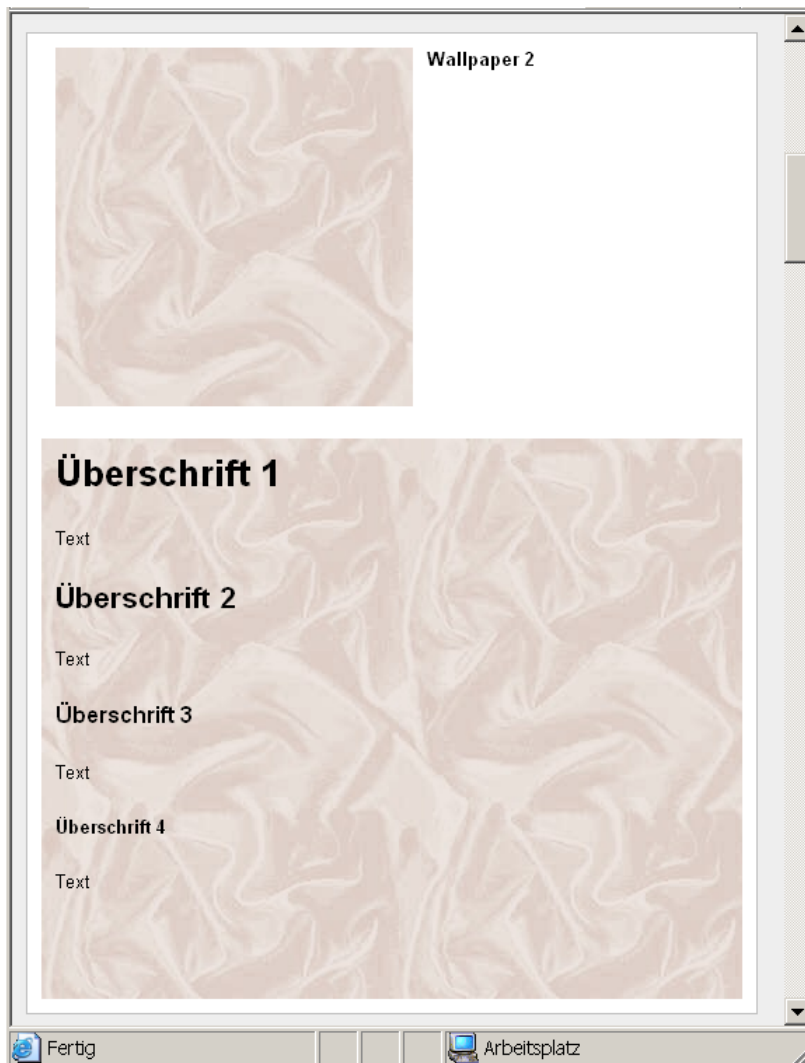
#### 24.2.4 Wallpapers (Hintergrundbilder)

Hintergrundbilder (Wallpapers) sind seit ihrer Einführung durch Netscape zu einem Dauerrenner im Web geworden. Wenn Sie nichts anderes angeben, wird ein Hintergrundbild so oft aneinander gereiht dargestellt, wie es erforderlich ist (Tapetenmuster- oder Kacheleffekt). Wenn die verwendete Grafik an den Seiten nicht einheitlich aussieht, entstehen sichtbare Kacheln. Das kann in manchen Fällen gewollt sein, in anderen aber zeugt es auch von Dilettantismus.

Es gibt mehrere Typen von Hintergrundbildern. Die klassischen und meistverwendeten stellen irgendeine informelle Struktur dar und bewirken auf diese Weise z.B. den visuellen Effekt einer faserigen, hölzernen, metallenen oder papiernen Oberfläche. Andere Hintergrundbilder stellen durch klare Farbflächen sichtbare Kacheln dar und bewirken dann im Kacheleffekt geometrische Muster. Und dann gibt es noch Hintergrundbilder vom Typ »Browser-Überlistung«. Das sind Grafiken, die extrem breit sind (z.B. 3000 Pixel), also bei üblichen Fensterbreiten im Browser nicht mehrmals nebeneinander angezeigt werden. In der Höhe haben sie dagegen oft nur 1 bis 5 Pixel. Die ersten 200 Pixel dieses langen Streifens enthalten dann beispielsweise ein Rechteck in einer bestimmten Farbe, während der gesamte Rest der Grafik eine andere Farbe hat. Im Browser-Fenster entsteht dann links ein Farbstreifen, über den sich beispielsweise Buttons zur Navigation platzieren lassen.

So pfiffig die Idee der Hintergrundbilder ist, so viel kann man dabei auch verkehrt machen. Bei allzu auffälligen Hintergrundbildern wird der Anwender zu sehr vom Vordergrund, dem eigentlichen Inhalt, abgelenkt. In vielen Negativbeispielen im Web leidet auch die Lesbarkeit der Textinformation unter dem Hintergrundbild. Generell gilt: Wenn Sie auf einer Web-Seite wichtigen Text anbieten wollen, sollte ein Hintergrund-Wallpaper angenehm, aber unauffällig

sein. Nur bei besonderen grafischen Effekten ist der Einsatz einer »aufregenden« Hintergrundgrafik gerechtfertigt.



24.4: Hintergrundmuster sollten nicht zu aufdringlich sein, damit der Text lesbar bleibt.

Um den gesamten Inhalt des Browser-Anzeigefensters mit einer Hintergrundgrafik zu unterlegen, wird die Hintergrundgrafik im einleitenden `<body>`-Tag der HTML-Datei eingebunden:

```
<body background="hintergrund.jpg" text="#000000">
```

Mit Hilfe der CSS-Eigenschaft `background-image` ist es außerdem möglich, auch anderen, untergeordneten HTML-Elementen einen Wallpaper-Effekt zuzuordnen.







24.6: Cliparts lockern textlastige Web-Seiten auf.

Manche Web-Seiten sind vollgepfropft mit irgendwo wahllos »besorgten«, fertigen Cliparts. Dabei werden meist zwei schwere Fehler begangen: Erstens sind zu viele verschiedene Grafiken an allen Ecken und Enden schlecht, weil jede einzelne Aufmerksamkeit erzwingt und beim Betrachter daher ein spontanes Gefühl von Stress, Überforderung und Orientierungslosigkeit entsteht. Und zweitens passen die Grafiken oft in Farbe, Größe und Stil nicht zueinander. Alles deutet darauf hin, dass da jemand wahllos eine der zahlreichen auf dem Markt erhältlichen CD-ROMs mit Clipart-Sammlungen ausgeschlachtet hat, aber ohne jedes Gespür für grafische Harmonie. Cliparts machen sich dann gut, wenn sie ein Gegengewicht zu vorhandenen Texten darstellen und einen nachvollziehbaren Bezug zum Text herstellen. Ideal sind Clipart-Serien, wo beispielsweise eine lustige Figur verschiedene Posen einnimmt und damit einen Textbezug herstellt.

Um Cliparts in HTML neben Text anzuordnen, bieten sich zwei Möglichkeiten an. Die eine ist, den Text um die Grafik fließen zu lassen (Kapitel 7.2). Das folgende typische Beispiel richtet eine Grafik rechts neben dem Text eines Absatzes aus, und der Text fließt um die Grafik:

```
<p>
Text des Absatzes</p>
```

Die andere Möglichkeit besteht darin, Text und Grafik mit Hilfe einer blinden Tabelle zu positionieren (Kapitel 8.6). Das folgende Beispiel zeigt das Prinzip:

```
<table border="0" cellspacing="10">
<tr><td valign="top">

```

```

</td><td valign="top">
Text links neben der Grafik
</td></tr></table>

```

### 24.2.7 Thumbnails (Vorschaugrafiken)

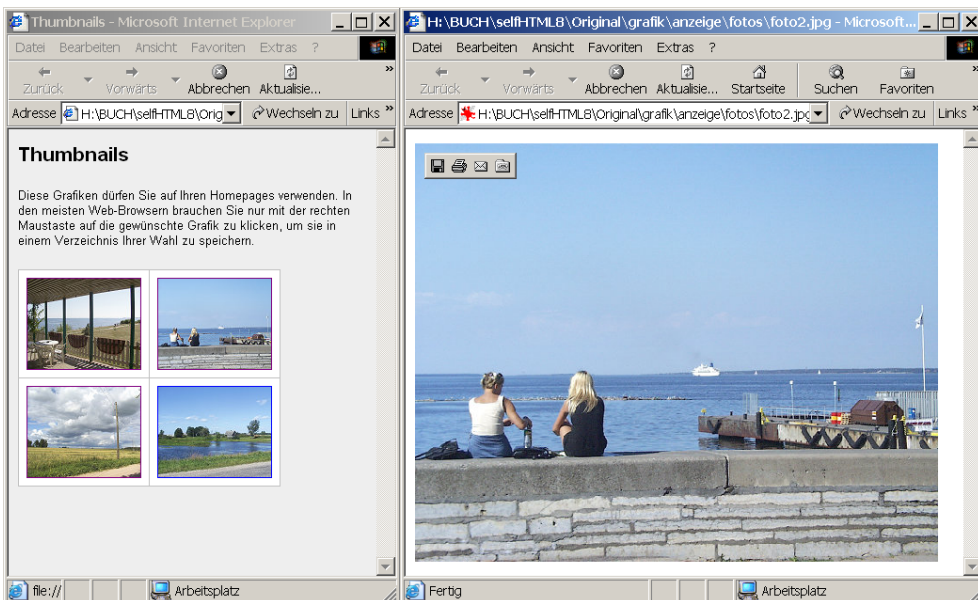
Leider stehen nicht allen Besuchern von Web-Seiten solche Bandbreiten der Datenübertragung zur Verfügung, die es erlauben, »mal eben« eine wunderschöne, aber speicherfressende Grafik am Bildschirm anzuzeigen. Wenn Sie großformatige Fotos oder Landkarten auf Ihren Web-Seiten anbieten, sollten Sie diese nicht »ohne Vorwarnung« in Seiten einbinden. Stattdessen sollten Sie eine stark verkleinerte Version der betreffenden Grafik einbinden. Die verkleinerte Version sollte gerade so groß sein, dass der Anwender entscheiden kann, ob er die Grafik in voller Größe betrachten möchte oder nicht. Die verkleinerte Grafik sollte anklickbar sein und beim Anklicken die große Grafik aufrufen, z.B. in einem neuen Anzeigefenster. Ein typisches HTML-Konstrukt dafür ist:

```

<a href="bild.jpg" target="_blank">

</a>

```



24.7: Beim Anklicken eines Thumbnails zeigt in diesem Beispiel ein zusätzliches Browser-Fenster das große Bild an.

### 24.2.8 Fotos

Viel zu viele Web-Seiten bieten viel zu wenig Fotos an. Dabei sind Menschen, also Besucher von Web-Seiten, doch von Natur aus so neugierig und freuen sich immer, wenn sie Fotos zu sehen bekommen. Im Zeitalter der preiswert gewordenen Digitalkameras und Scanner ist es auch wirklich kein großes Problem mehr, eigene Aufnahmen als Pixelgrafiken in den Computer zu bekommen. Was Sie dann nur noch brauchen, ist ein Grafikprogramm, mit dem Sie die Bilder gegebenenfalls auf eine webgerechte Größe bringen und in einem geeigneten Grafik-Format wie JPEG abspeichern können. Bei den meisten Scannern und Digitalkameras gehören solche Programme als Zugabe zum Produkt.

Fotos auf Web-Seiten brauchen auch gar nicht sonderlich großformatig sein. Im Gegenteil – das bricht meistens nur das Seitenlayout auf, sorgt für lange Ladezeiten beim Anwender und sieht meistens nicht mal besonders gut aus. Bilder von Digitalkameras, die mehrere »Megapixel« Auflösung anbieten, sollten also erst mal in der Größe reduziert werden. Wichtig ist dabei, dass volle 16,7 Mio. Farben eingestellt sind. Gute Grafikprogramme besitzen ausgereifte Algorithmen, sodass eine Größenreduzierung dem Erscheinungsbild des Fotos keine große Gewalt antut. Wichtig ist bei Fotos, dass die Größenreduktion über die Angabe von gewünschten Pixeln für Breite und Höhe erfolgt, nicht über Angaben aus dem Print-Bereich, wie Zentimeter oder Zoll. Für ansprechende Bilder auf Web-Seiten reichen meist 200 bis 400 Pixel für die kürzere Seite und 300 bis 600 Pixel für die breitere Seite. Dabei entstehen bei JPEG und mittlerer Kompression auch akzeptable Dateigrößen, die in der Regel unter 100 Kilobyte liegen.

### 24.2.9 Blinde Pixel

HTML-Puristen bekommen einen nervösen Hautausschlag, wenn sie von »blinden Pixeln« hören. Aber besser als diesen Trick totzuschweigen ist es, ihn zu verraten und darauf hinzuweisen, warum und wieso er nicht selten eingesetzt wird und welche Probleme damit verbunden sein können.

Der blinde Pixel ist eine Erfindung von David Siegel, einem der frühen Gurus des Web-Designs. Verärgert über das Verhalten bei Web-Browsern, Tabellen nicht mit den in HTML notierten Attributwerten für `width=` und `height=` darzustellen, wenn der Zelleninhalt schmaler ist und Nachbarzellen relativ viel Inhalt haben, erzwang er den gewünschten Raum einfach mit Hilfe einer unsichtbaren Grafik.

Der Trick besteht aus zwei Teilen. Im ersten Teil wird eine sehr kleine GIF-Grafik erzeugt, die überhaupt nichts darstellt, sondern nur aus einer einfarbigen Fläche besteht. Es ist besser, wenn die Grafik nicht wirklich nur 1 mal 1 Pixel groß ist, sondern ca. 5 mal 5 Pixel. So werden Probleme bei einigen Browsern vermieden. Die eine Farbe, aus der die GIF-Grafik besteht, wird so dann als transparent definiert. Der Effekt ist, dass die Grafik Raum beansprucht, aber nicht sichtbar ist. Darunter liegende Farben, etwa definierte Hintergrundfarben von Tabellenzellen, in denen die unsichtbare Grafik eingebunden wird, scheinen durch.

Der zweite Teil des Tricks besteht in der Möglichkeit, beim Einbinden der unsichtbaren Grafik die Attribute `width=` und `height=` im `<img>`-Tag auf beliebige gewünschte Werte zu setzen. Die Grafik wird dann zwar vom Browser verzerrt, aber da sie ja unsichtbar ist, sieht man auch nichts von der Verzerrung. So lassen sich mit der einen Grafik, die nur ein paar Dutzend Byte groß ist, beliebige horizontale und vertikale Räume und Abstände erzwingen.

Der folgende HTML-Ausschnitt zeigt eine typische Anwendung. Die Grafik wird in zwei nebeneinander liegenden Tabellenzellen gleich zu Beginn referenziert. Ihre Höhe wird in beiden Referenzen auf 1 Pixel verzerrt, und ihre Breite auf den jeweils gewünschten Wert. Auf diese Weise erhält die Tabelle zwei Spalten mit unumstößlichen Mindestbreiten:

```
<table border="0" cellspacing="0" cellpadding="0">
<tr><td valign="top">

Hier folgt der Inhalt der ersten Spalte, die mindestens 200 Pixel breit ist.
</td><td valign="top">

Hier folgt der Inhalt der zweiten Spalte, die mindestens 400 Pixel breit ist.
</td></tr></table>
```

Unsauber ist die Methode mit dem blinden Pixel insofern, als hässliche »broken-image«-Symbole angezeigt werden, wenn die Grafik beim Übertragen der Web-Seite an den Browser nicht übertragen wird, z.B. weil der Anwender den Ladevorgang abbricht oder das Anzeigen von Grafiken abgeschaltet hat. Darüber hinaus wird es von Puristen als Zweckentfremdung von Grafik und HTML-Eigenschaften empfunden, solche Mittel einzusetzen. Und schließlich ist das Erzwingen fester Breiten ohnehin zweifelhaft, da der Anwender dann beispielsweise bei der Anzeige in nicht genügend großen Browser-Fenstern gezwungen ist, quer zu scrollen, was nicht besonders anwenderfreundlich ist. Andererseits ist der blinde Pixel ein Notbehelf, um fehlende Möglichkeiten in HTML und CSS auszugleichen. Mittlerweile sind zwar bestimmte CSS-Eigenschaften zur Positionierung von Elementen, die den Zweck des blinden Pixels ebenfalls erfüllen, verfügbar, aber nur in neueren Browsern.

### 24.2.10 Grafische Layout-Elemente

Grafiken dienen beim Web-Design nicht selten dazu, Dinge zu realisieren, die mit HTML und CSS (noch) nicht möglich sind. Dazu gehören beispielsweise Rundbögen. Solche grafischen Elemente empfindet der unbedarfte Betrachter einer Web-Seite nicht als Grafiken »auf der Web-Seite«, sondern als Basisbestandteile der Formgebung. Ähnlich wie bei blinden Pixeln fallen solche Grafiken dann negativ auf, wenn sie beim Laden der Seite nicht übertragen werden und ein »broken-image« in der Anzeige erzeugen.

Eingesetzt werden solche grafischen Bestandteile meistens in blinden Tabellen oder absolut positionierten Elementen. Durch nahtloses Aneinanderfügen von Tabellenzellen oder positionierten Elementen, die als Inhalt eine entsprechende Grafik haben, lassen sich solche Layout-Effekte wie die im Anzeigebeispiel erzielen. Bei Verwendung von blinden Tabellen für solche Effekte ist es wichtig, dass die Werte für die Attribute `border=`, `cellspacing=` und `cellpadding=` explizit auf 0 gesetzt werden. Ein typisches HTML-Konstrukt sieht etwa so aus:

```
<table border="0" cellspacing="0" cellpadding="0"><tr>
<td></td>
<td></td>
</tr><tr>
<td></td>
<td><!-- Inhalt wie Text, andere Grafiken - oft auch "innere Tabellen" usw. --></td>
</tr></table>
```



24.8: Viele einzelne Grafikelemente bilden dieses Layout einer Reise-Site.

## 24.3 Techniken und Begriffe der Bildbearbeitung

### 24.3.1 Allgemeines zu Raster- bzw. Pixelgrafiken

Bildbearbeitung ist eine Welt für sich und kann im Rahmen der vorliegenden Dokumentation nur gestreift werden. Die Bearbeitung von Grafiken ist auch immer noch stark an die Fähigkeiten einzelner Softwareprodukte gebunden. Wenn Sie Grafiken für Web-Seiten selbst erstellen oder bearbeiten wollen, brauchen Sie deshalb auf jeden Fall ein professionelles oder zumindest semi-professionelles Grafikprogramm für Pixelgrafik. Betriebssystem-Bordmittel wie Paint unter Windows reichen dazu nicht aus. Das führende Produkt zur Bildbearbeitung ist wohl nach wie vor Photoshop. Andere Programme, die deutlich preiswerter oder sogar Freeware sind und sich im Vergleich zu Photoshop aber nicht verstecken brauchen, sind ebenfalls erhältlich. Im Link-Verzeichnis des Online-Angebots von SELFHTML ([selfaktuell.teamone.de](http://selfaktuell.teamone.de)) aktuell finden Sie eine Softwareübersicht.

Um die Funktionen solcher Grafikprogramme richtig einzusetzen, benötigen Sie Kenntnisse. Ebenfalls im Link-Verzeichnis des Online-Angebots von SELFHTML aktuell finden Sie Verweise zu Grafik-Tutorials im Web.

Einige wichtige Begriffe und Funktionen der Bildbearbeitung behandelt auch dieser Abschnitt. Dabei geht es um die Technik der Rasterbilder. Rasterbilder sind einfach alle Bilder, deren Inhalt nicht durch vektorielle Angaben beschrieben sind, sondern durch Raster von Bildpunkten. Ein für Bildschirmgrafiken optimales Raster sind Bildschirmpixel. Pixelgrafiken sind also eine Form von Rasterbildern. Die beiden heute verbreitesten Formate für Web-Grafiken, das GIF-Format und das JPEG-Format, sind beide auf das Abspeichern solcher Pixelgrafiken spezialisiert.

### 24.3.2 Bildgröße, dpi und Pixel

Wenn Sie eine Pixelgrafik neu erstellen, müssen Sie zunächst die Bildgröße durch Angabe von Breite und Höhe der Grafik festlegen. Eine Pixelgrafik (oder Rastergrafik) besteht – im Gegensatz zu einer Vektorgrafik – aus einer Anzahl von Bildpunkten, genannt Pixel. Die Breite einer solchen Grafik ergibt sich dann aus der Anzahl der Pixel, die horizontal in einer Reihe liegen, und die Höhe aus der Anzahl der Pixel, die vertikal in einer Reihe liegen. Solange alles am Bildschirm bleibt, geht diese einfache Rechnung auf. Und solange Sie Pixelgrafiken nur für den Einsatz auf Web-Seiten benötigen, die am Bildschirm angezeigt werden, brauchen Sie sich über andere Probleme der Bildgröße auch keine Gedanken machen.

Die Probleme entstehen erst, wenn eine Pixelgrafik auf einem anderen, nicht pixelorientierten Medium ausgegeben werden soll. Bei der Ausgabe auf Papier etwa, also beim Drucken, muss festgelegt werden, wie »groß« ein Pixel auf dem Papier sein soll, denn im Gegensatz zum Bildschirm gibt es auf Papier keine vorgegebenen Pixel. Beim Druck gibt es druckbare Punkte. Je dichter sie zusammen liegen, desto feiner die Auflösung des ausgedruckten Bildes. Diese Dichte wird in Dots per Inch (dpi) gemessen.

Da nun viele Pixelgrafiken für den Druck gedacht sind, bieten die meisten Grafikprogramme beim Festlegen der Bildgröße zwei oder mehrere Varianten an. Die eine besteht darin, Breite und Höhe des Bildes in Pixeln anzugeben, die andere darin, eine dpi-Dichte zu wählen oder die Größe des Bildes in Maßeinheiten wie Zentimetern oder Zoll bzw. Inch zu bestimmen. Die zweite Variante ist auf jeden Fall nur für die Weiterverarbeitung der Grafik für den Druck interessant. Um die Größe eines Bildes für die Bildschirmanzeige zu bestimmen, sollten Sie stets die Variante wählen, die Bildgröße über die Angabe von Pixeln für Breite und Höhe zu bestimmen.

Bereits vorhandene Pixelgrafiken haben entweder schon die Größe am Bildschirm, in der Sie sie für den Einbau in Ihre Web-Seiten brauchen können, oder die vorhandenen Grafiken sind am Bildschirm zu groß oder zu klein.

Wenn das Bild zunächst viel größer ist, als Sie es für die Bildschirmanzeige benötigen, müssen Sie es durch Festlegen von geringeren Pixelwerten für Breite und Höhe verkleinern. Erforderlich ist das beispielsweise bei Fotos von Digitalkameras, die oft mehrere Megapixel groß sind und am Bildschirm bei 1:1-Betrachtung riesig wirken. Auch Grafiken, die durch Einscannen bei hoher dpi-Zahl entstanden sind, sind so groß. Zum Herunterrechnen der Größe eines Bildes besitzen moderne Grafikprogramme ausgereifte Algorithmen. Einige bieten zum so genannten »Resample«, also zum Neuberechnen des Bildinhaltes für eine andere Bildgröße, auch mehrere Algorithmen zur Auswahl an. Zwei Faktoren sind zum Resample auf jeden Fall wichtig: Es sollte die volle

Farbtiefe (z.B. 16,7 Millionen Farben) eingestellt sein, und es sollte proportional verkleinert werden, d.h. die Seitenverhältnisse sollten beibehalten werden, damit das Grafikprogramm die Grafik beim Verkleinern nicht auch noch zusätzlich verzerren muss.

Wenn das Bild am Bildschirm kleiner ist, als Sie es haben möchten, sieht die Sache schlecht aus. Durch »Resample« können Sie eine Grafik zwar nicht nur verkleinern, sondern auch vergrößern. Doch die Ergebnisse sind dabei oft unbefriedigend. Denn es ist leichter, aus viel Information weniger zu machen (wie beim Verkleinern), als aus wenig Information mehr (wie beim Vergrößern). In solchen Fällen ist es meist besser, sich mit der zu kleinen Bildgröße abzufinden.

### 24.3.3 Farbtiefen, Farbmodelle und Farbkanäle

Die Farbtiefe einer Grafik definiert, wie viele verschiedene Farbtöne in der Grafikdatei gespeichert werden können. Bei Pixelformaten gibt es folgende typischen Farbtiefen:

- 2 Farben (schwarz/weiß)
- 16 Farben
- 256 Farben
- 16,7 Mio. Farben

Das GIF-Format unterstützt 2, 16 und 256 Farben, das JPEG-Format 16,7 Mio. Farben, ebenso wie das PNG-Format. Dass in einer Datei so viele Farben gespeichert werden können, bedeutet aber noch lange nicht, dass Anwender tatsächlich so viele Farben sehen. Denn wie viele Farben beim Anwender angezeigt werden können, hängt von der eingesetzten Hardware (Grafikkarte, Bildschirm) ab. Nur die wenigsten Anwender verfügen über Hardware, mit deren Hilfe volle 16,7 Mio. Farben angezeigt werden können. Viele heutige Standard-PCs bieten Farbtiefen zwischen 16.000, 32.000 oder 64.000 Farben an.

Bei Pixelgrafiken wird für jeden Pixel dessen Farbe gespeichert. Um die Farben in ein computer-gerechte Darstellungsform zu bringen, müssen sie als Zahlenwerte dargestellt werden. Das ist ganz ähnlich wie bei der Umsetzung von Zeichen in Bits und Bytes. Bei der numerischen Repräsentation von Farben im Computer gibt es verschiedene Modelle. Die beiden bekanntesten Modelle sind:

- **Das RGB-Modell:** Beim RGB-Modell wird eine Farbe durch ihre Anteile an den drei Grundfarben Rot, Grün und Blau definiert. Jede Farbe hat also einen Rotwert, einen Grünwert und einen Blauwert. Jeder der drei Werte wird durch Zahlen zwischen 0 und 255 definiert. Der Wert 0 bedeutet: keinen Anteil an der betreffenden Grundfarbe, der Wert 255 bedeutet: maximalen Anteil an der betreffenden Grundfarbe. Ein dunkles Blau hat nach diesem Schema z.B. die Farbwerte 0,0,153. (0 rot, 0 grün, 153 blau). Mit diesem Schema können bis zu 16,7 Millionen unterschiedliche Farben definiert werden. Eine schematische Darstellung der Farbmischung finden Sie auf der Farbtabelle im Anhang.
- **Das HSB-Modell:** Auch bei diesem Modell gibt es drei definierende Zahlenwerte. **H** steht für *Hue* (Farbton), **S** für *Saturation* (Sättigung), und **B** für *Brightness* (Helligkeit). Farbtöne können Werte zwischen 0 und 359 annehmen, Sättigung und Helligkeit Werte zwischen 0 und 255. Auf diese Weise lassen sich ca. 23,6 Mio. Farben definieren. Eine typische Dialogbox

eines Grafikprogramms für das Definieren von HSB-Farbtönen finden Sie auf der Farbtafel im Anhang.

Für Web-Grafiken ist das RGB-Modell maßgeblich. Wenn Sie zum Erstellen von Web-Grafiken mit einem Grafikprogramm arbeiten, das mehrere Farbmodelle kennt, halten Sie sich an das RGB-Modell.

### 24.3.4 Farbpaletten und indizierte Farben

Im GIF-Format können Sie pro GIF-Datei aus den 16,7 Mio. möglichen Farben des RGB-Farbmodells maximal 256 Farben »aussuchen« und abspeichern. Bei einer solchen Farbauswahl spricht man von einer Farbpalette.

Jede der 256 Farben kann beliebige RGB-Werte haben, d.h. es sind auch »krumme« Farbwerte wie 71,217,34 möglich. Für webgerechte GIF-Grafiken steht jedoch ein Standardschema zur Verfügung, das von Netscape eingeführt wurde und auf diesem Browser plattformübergreifend optimal interpretiert wird. Nach dem Netscape-Schema sind alle Farben erlaubt, deren RGB-Werte durch 51 dividierbar sind. R(ot), G(rün) und B(lau) sollten danach also einen der Werte 0, 51, 102, 153, 204 oder 255 (hexadezimal: 00, 33, 66, 99, CC oder FF) haben. Ein erlaubter Wert ist danach z.B. 255,51,204 (hexadezimal: FF,33,CC). Auf diese Weise ergeben sich 6 x 6 x 6 mögliche Farben, also 216. Die verbleibenden 40 Farben können Sie als »eiserne Reserve« betrachten. Wenn Sie für einzelne Grafiken bestimmte, fehlende Farbtöne brauchen, können Sie unbenutzte Farben der Palette mit der gewünschten Farbe definieren. Eine Darstellung der Farbpalette finden Sie im Anhang.

Wenn Sie Ihre Ansprüche an die Hardware des Endanwenders noch weiter zurückschrauben wollen, können Sie sich auch auf Grafiken beschränken, die eine 16-Farben-Palette enthalten. Auch dabei gilt: Theoretisch kann jede der 16 Farben einer solchen Farbpalette beliebige Farbwerte haben. Es gibt jedoch 16 Farben, die von jedem VGA-kompatiblen Bildschirm angezeigt werden können. In MS Windows sind das deshalb auch die 16 Grundfarben. Bei einer Beschränkung auf 16 Farben ist es sinnvoll, mit diesen 16 Grundfarben zu arbeiten.

### 24.3.5 Alphakanal und echte Transparenz

Der Alphakanal ist eine Erweiterung der Farbmodelle. Dabei wird für jeden Bildpunkt (Pixel) neben dessen Farbwert ein Alphawert gespeichert. Beim RGB-Modell spricht man daher auch vom erweiterten RGBA-Modell (Rot-Grün-Blau-Alpha). Über den Alphakanal wird die Transparenz der Farbe beschrieben.

Sinnvoll ist das Arbeiten mit dem Alphakanal, wenn man mit der so genannten Layer-Technik arbeitet. Dabei wird eine Grafik durch Übereinanderlegen mehrerer Ebenen komponiert, wobei jede Ebene ein oder mehrere bestimmte Bestandteile der Grafik enthält. Wenn Sie nichts anderes angeben, überdecken sich die Ebenen dabei. Mit Hilfe diverser Filtereffekte, die im Alphakanal gespeichert werden, lässt sich jedoch bestimmen, dass und wie eine unterhalb angeordnete Deckfarbe durch eine oberhalb angeordnete Deckfarbe hindurchschimmert.





24.9: Die beiden Abbildungen zeigen zwei typische Filtereffekte, die durch Arbeit mit mehreren Ebenen und Alphakanal entstanden (farbige Darstellung am Buchende).

In diesem Fall wurde ein Foto als erster Layer (erste Ebene) definiert, und ein Schriftzug NATUR als zweiter Layer. Der zweite Layer enthält nur den Schriftzug und ist ansonsten vollständig transparent. Damit der untere Layer aber auch durch den Schriftzug hindurchschimmern kann, ist der Alphakanal erforderlich. Bei aktiviertem Alphakanal lassen sich die gewünschten Filtereffekte einstellen.

Gute Grafikprogramme unterstützen Sie bei der Arbeit mit mehreren Layern und mit dem Alphakanal. Bei den Transparenz-Effekten, wie sie mit Hilfe des Alphakanals möglich sind, spricht man auch von »echter« Transparenz. Im Gegensatz dazu steht die im folgenden beschriebene Transparenz einer bestimmten Palettenfarbe.

### 24.3.6 Transparenz einer bestimmten Palettenfarbe

Die meisten Web-Browser bieten dem Anwender per Voreinstellung einen hellgrauen oder weißen Hintergrund für das Anzeigefenster an, sehen jedoch die Möglichkeit vor, die Hintergrundfarbe dem eigenen Geschmack anzupassen. Wenn Sie nun eine Grafik mit hellgrauem Hintergrund haben, wirkt diese Grafik auf einem weißen oder bunten Fensterhintergrund wie ein »hässlicher« grauer Fleck. Ideal ist da eine Grafik, die einen unsichtbaren (transparenten) Hintergrund hat, sodass sie auf jedem Fensterhintergrund wirkt. Auch in Verbindung mit selbst definierten Hintergrundfarben und Hintergrundbildern (Wallpapers) ist der Einsatz von Grafiken mit transparentem Hintergrund sinnvoll.

Das GIF-Format sieht die Möglichkeit vor, eine bestimmte Palettenfarbe als unsichtbar zu definieren. Dies ist jedoch erst mit dem neuen 89er-Format von GIF möglich, im älteren, 87er-Format noch nicht. Sinnvoll ist die Anwendung bei Grafiken, die einen klaren, einfarbigen Hintergrund haben. Moderne Grafikprogramme unterstützen beim Auswählen und Testen der transparenten Palettenfarbe.

### 24.3.7 Dithering (Error-Diffusion)

Wenn Sie mit Farbpaletten und indizierten Farben arbeiten und so ein begrenztes Farbschema auf Bilder mit feinen Übergängen und zahlreichen Zwischentönen anwenden, dann muss der Computer eine Lösung für das Problem finden, dass weniger Farbwerte zur Verfügung stehen,

als das Bild erfordert. Dazu gibt es verschiedene Algorithmen. Die folgende Abbildung zeigt zwei Möglichkeiten, darunter diejenige mit dem Algorithmus für Dithering:



Farbverlauf mit 16,7 Millionen Farben (am Buchende finden Sie eine Farbdarstellung der vier Bilder)

24.10:



Bild auf Farbpalette mit 16 optimierten Indexfarben reduziert **ohne** Dithering!

24.11:

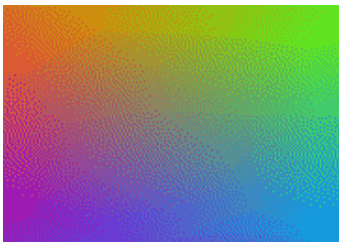
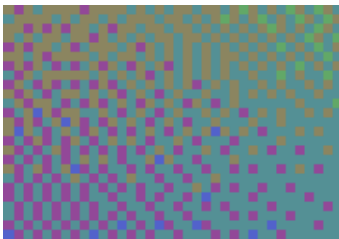


Bild auf Farbpalette mit 16 optimierten Indexfarben reduziert **mit** Dithering!

24.12:



Ausschnittvergrößerung, die den Dithering-Effekt deutlich macht.

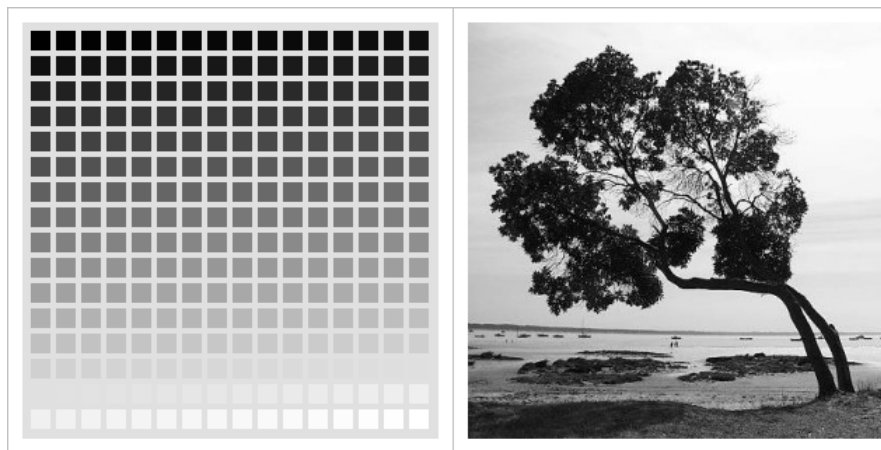
24.13:

Beim Dithering werden Farbverläufe in Punktmuster aufgelöst, die mit Hilfe der vorhandenen Palettenfarben darstellbar sind. Das Prinzip wird auch als Error-Diffusion bezeichnet. Der bekannteste Algorithmus dafür ist der Floyd-Steinberg-Algorithmus, der in den meisten Grafikprogrammen zum Einsatz kommt. Beachten Sie, dass GIF-Grafiken mit intensivem Dithering schnell sehr groß werden können. Meistens ist es besser, solche Grafiken gleich als JPEG-Grafi-

ken abzuspeichern. Dann muss allerdings der Web-Browser des Anwenders das Dithering besorgen, falls die Grafikumgebung, auf der er läuft, nur eine begrenzte Anzahl von Farben zulässt.

### 24.3.8 Graustufen

Bei Graustufenbildern wird typischerweise mit einer Farbpalette von 256 Farben gearbeitet. Es handelt sich jedoch um eine ganz bestimmte Palette. Rot- Grün- und Blau-Anteile sind bei allen 256 definierten Werten jeweils gleich, was in der Addition zu einem Ton zwischen Schwarz und Weiß führt. Wenn R, G und B den Wert 0 haben, ist die damit definierte Farbe ein reines Schwarz. Wenn alle drei Farbanteile den Wert 255 haben, wird damit ein reines Weiß definiert. Ein Zwischenwert, etwa 192,192,192, definiert eine graue Farbe von bestimmter Helligkeit. Die Palette eines Graustufenbildes besteht nun darin, dass darin alle 256 Farben zwischen 0,0,0 und 255,255,255 gespeichert sind – also 256 Grautöne.



24.14: Die Abbildung zeigt links die typische Graustufentabelle, bestehend aus 256 Grautönen von Schwarz bis Weiß, und rechts die Anwendung einer solchen Graustufentabelle auf ein Foto.

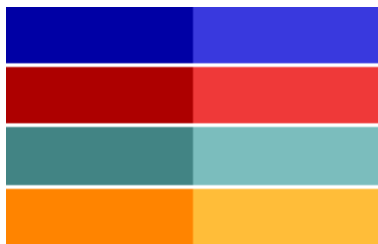
In der Beispielabbildung wurde allerdings das Foto nach der Reduzierung auf Graustufen wieder auf volle Farbtiefe gesetzt und dann im JPEG-Format, nicht im GIF-Format abgespeichert. Die Grafik wird dadurch deutlich kleiner. Denn auch bei der Reduzierung eines Farbfotos mit voller Farbtiefe auf Graustufen wird kräftig mit Dithering gearbeitet, was die Dateigrößen bei Fotos schnell in die Höhe treibt.

### 24.3.9 Helligkeit, Sättigung und Kontrast

Helligkeit, Sättigung und Kontrast sind typische Funktionen der Bildbearbeitung, die sich auf ein ganzes Bild oder auf zuvor markierte Ausschnitte anwenden lassen.

### **Helligkeit**

Farben können den gleichen Farbton haben, aber unterschiedlich hell wirken. Die folgenden Farbvergleiche zeigen jeweils links eine Farbe und rechts eine hellere Variante davon. Sie finden eine farbige Variante der folgenden Bilder auf der Farbtafel im Anhang.

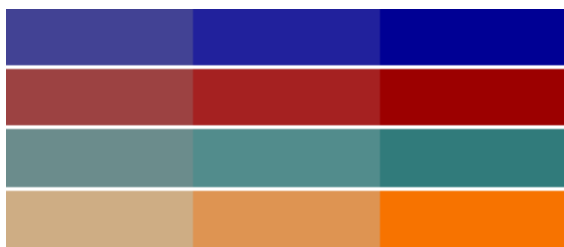


24.15: Helligkeit

Für die Farben der rechten Seite wurden ca. 25% mehr Helligkeit gegenüber den Farben der linken Seite eingestellt. Durch Helligkeit werden Farben **nicht** leuchtender! Kräftige Farben bleiben kräftig, Pastellfarben bleiben pastell.

### **Sättigung**

Im Gegensatz zur Helligkeit beeinflusst die Sättigung die subjektiv empfundene Leuchtkraft einer Farbe. Die folgenden Farbvergleiche zeigen in der Mitte jeweils eine Ausgangsfarbe, links davon eine Variante mit geringerer Sättigung, und rechts davon eine Variante mit höherer Sättigung.



24.16: Sättigung

Was die Helligkeit betrifft, werden die Farbtöne jeweils als etwa gleich hell eingestuft. Farben mit geringerer oder höherer Sättigung wirken zwar beim ersten Hinsehen oft heller oder dunkler, aber das stellt sich als Trugschluss heraus.

### **Kontrast**

Eine Erhöhung des Kontrasts bewirkt, dass dunkle Farben noch dunkler und helle Farben noch heller werden. Die Unterschiede zwischen hellen und dunklen Farben werden also betont. Besonders bemerkbar macht sich das bei Fotos beispielsweise an scharfen Farbübergängen, etwa bei Häuserecken, wo eine Seite von der Sonne beschienen wird und die andere im Halbschatten liegt. Wird der Kontrast dagegen verringert, werden helle Farben dunkler und dunkle heller. Die

Farben des Bildes gleichen sich also stärker an. Harte Farbübergänge erscheinen weicher, das Bild wirkt insgesamt grautöniger. Die folgenden Farbvergleiche zeigen in der Mitte jeweils eine Ausgangsfarbe, links davon eine Variante mit niedrigerem Kontrast und rechts davon eine Variante mit höherem Kontrast.



24.17: Kontrast



24.18: Stark erhöhter Kontrast

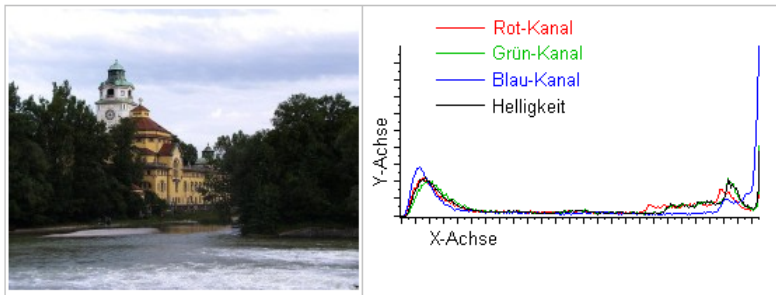
Bei stark erhöhtem Kontrast gehen Feinheiten im Farbverlauf verloren, es kommt zum Schlagschatten-Effekt. Bei stark reduziertem Kontrast entsteht ein Gauschleier-Effekt. Die folgenden drei Bilder zeigen in der Mitte ein Ausgangsbild, links reduzierten Kontrast (Gauschleier) und rechts erhöhten Kontrast (Schlagschatten).

Zusätzlich zur einfachen Kontrastkorrektur bieten die meisten besseren Grafikprogramme eine farblich abgestufte Tonwertkorrektur an, mit deren Hilfe sich Grobheiten, wie sie bei starker Kontrastveränderung entstehen, vermeiden lassen.

### 24.3.10 Histogramm

Für jede Pixelgrafik lässt sich ein Histogramm erstellen. Das ist die grafische Darstellung der Farb- und Helligkeitsverteilung in der Grafik. Zur Darstellung dient dabei ein Koordinatensys-

tem mit einer X-Achse (horizontal) und einer Y-Achse (vertikal). Die X-Achse bedeutet die Helligkeit. Der Achsenursprung steht für maximale Dunkelheit. Je weiter die Achse reicht, desto mehr Helligkeit bedeutet dies. Die Y-Achse steht für die Anzahl der Pixel im Bild. Im Koordinatensystem lässt sich nun ablesen, wie viele Pixel welchen Farbwert bzw. welche Helligkeit haben.



24.19: Ein Foto und sein Histogramm.

Das Histogramm zeigt in diesem Beispiel zweierlei: erstens zwei auffällige Spitzen, eine im sehr dunklen Bereich und eine im sehr hellen Bereich; zweitens zeigt es, dass die Rot-, Grün- und Blau-Kurven des Bildes vergleichsweise parallel verlaufen. Letzteres ist ein Indiz dafür, dass die Farbverteilung einigermaßen in Ordnung ist und das Bild keinen Farbstich in eine bestimmte Richtung hat. Die beiden Spitzen im unteren und im obersten Helligkeitsbereich deuten jedoch auf eine starke Dominanz sehr heller und sehr dunkler Farben hin, was ja auch ersichtlich ist.

Gute Grafikprogramme erlauben das Bearbeiten der grafischen Histogrammkurven. Auf diese Weise lässt sich die Farbverteilung im Bild korrigieren. Im obigen Beispiel wäre es ein Versuch wert, die Spitze im unteren Bereich zu reduzieren, also die RGB-Werte dort näher an den Ursprung der Y-Achse zu bekommen. Natürlich macht das nur Sinn, wenn die damit verbundene Bildaufhellung bei den dunklen Tönen keine hässlichen Verfälschungen bewirkt. Bildkorrekturen auf Histogramm-Ebene erfordern deshalb viel Geduld und Ausprobieren.

### 24.3.11 Gammakorrektur

Bei der Gammakorrektur ist es zunächst wichtig zu wissen, was ein Gammawert ist. Ein Gammawert ist bei Computergrafiken das Verhältnis zwischen den Farbwerten, die in der Grafik gespeichert sind, und den Farbwerten, die ein Gerät wie Bildschirm oder Drucker daraus macht. Idealerweise wäre das ein 1:1-Verhältnis, sprich, es werden genau die Farben ausgegeben, die in der Grafikdatei gespeichert sind. Das ist aber nicht der Fall. Bildschirme beispielsweise sind leider nicht »geeicht«. Die Darstellung von Farben schwankt von Bildschirm zu Bildschirm und von Betriebssystem zu Betriebssystem sogar gewaltig. Wenn Sie also an Ihrem Bildschirm eine Grafik bearbeiten und mit Funktionen wie Helligkeit, Sättigung, Histogramm usw. deren optische Wirkung beeinflussen, dann gilt das Ergebnis zunächst mal nur für Ihren Bildschirm. Wenn Sie die Grafik im Web präsentieren wollen, wird sie jedoch auf sehr vielen und sehr unterschiedlichen Bildschirmen angezeigt.

Gute Grafikprogramme bieten die Gammakorrektur deshalb auf zwei Ebenen an: auf der Ebene des eigenen Monitors und auf der Ebene konkreter Grafiken. Die Gammakorrektur für den Monitor leistet eine Anpassung zwischen numerischen Farbwerten, etwa nach dem RGB-Modell,

und deren Darstellung im Grafikprogramm an Ihrem Bildschirm. Die Bildschirmgrundeinstellungen selber werden dabei nicht verändert. Die Korrektur betrifft lediglich die Richtigstellung der Farbanzeige im Grafikprogramm. Eine grafische Darstellung erlaubt bei solchen Grafikprogrammen die Angleichung zwischen gespeicherten und am Bildschirm dargestellten Farbwerten. Bei den meisten Computerbildschirmen müssen Sie einen Gammawert von 1,5 oder höher einstellen, um eine realistische Farbdarstellung im Grafikprogramm zu erhalten. Wenn jeder, der Grafiken bearbeitet, das tun würde, wäre insgesamt schon viel gewonnen.

Nachdem die Gammakorrektur für den Monitor erfolgt ist, lassen sich mit Hilfe der Gammakorrektur für konkrete Grafiken Farbprobleme ausgleichen, die durch die Herkunftsquelle der Grafik bedingt sind. So nehmen beispielsweise viele Digitalkameras Bilder scheinbar zu dunkel auf. Das liegt daran, dass auch bei der Umrechnung von Lichtstärke in RGB-Werte, die eine Digitalkamera bei einer Aufnahme vornehmen muss, Ungleichheiten vorkommen, die von Kamera zu Kamera schwanken können. Durch eine Gammakorrektur lassen sich solche Probleme meistens gut lösen.



24.20: Das linke der beiden (folgenden) Bilder zeigt ein Ausgangsbild, das rechte das Ergebnis nach einer Gammakorrektur (Gammawert 1,8).

Bei einem Gammawert größer als 1,0 werden vor allem die mittleren Farbtöne gegenüber den sehr dunklen und sehr hellen Farbtönen aufgehellt. Bei einem Gammawert kleiner als 1,0 werden diese Farbtöne abgedunkelt. Der Effekt ist dabei ein anderer, als wenn Sie etwa versuchen, durch Erhöhen der Helligkeit eine Grafik aufzuhellen.

### 24.3.12 Filtereffekte

Jede Form der Manipulation einer Pixelgrafik, z.B. Helligkeit, Kontrast, Sättigung oder Gammakorrektur, ist ein Filter. Denn auf Rechnerebene ist das Ändern von Helligkeit, Kontrast usw. mit einem bestimmten mathematischen Algorithmus verbunden, der auf die gespeicherten RGB-Werte in der Grafik angewendet wird. Diese werden durch die Anwendung des Algorithmus neu berechnet. Das Ergebnis ist dann das geänderte Aussehen des Bildes aufgrund der neuen RGB-Werte der einzelnen Pixel.



Solange Sie mit den Standardfunktionen der Bildmanipulation arbeiten, brauchen Sie deren Filter-Algorithmen nicht kennen. In den Dialogen, die das Grafikprogramm anbietet, verändern Sie lediglich Parameter, aber nicht den zugrunde liegenden Algorithmus. So stellen Sie beispielsweise den gewünschten Wert für Sättigung ein, aber Sie wissen nicht, wie das Grafikprogramm eigentlich rechnet, um aus dem eingestellten Wert die RGB-Werte der einzelnen Pixel neu zu berechnen.

Gute Grafikprogramme bieten eine ganze Reihe von weiteren Filtern im Dialog an. Dazu gehören Standards wie Relief-Effekte, Rausch- und Verwisch-Effekte usw. Daneben bieten einige Programme auch eine so genannte Plug-In-Schnittstelle für Filter an. Diese erlaubt es, sich beliebige eigene Filter schreiben, die einen Algorithmus auf die RGB-Werte einer Grafik bewirken. Ein Algorithmus könnte beispielsweise lauten: Ziehe von allen Rot-Werten, die über 200 liegen, 50 ab, von allen Grün-Werten, die über 200 liegen, 100, und von allen Blau-Werten, die über 200 liegen, 150. Oder: Setze bei allen Pixeln, bei denen der Rot-Wert, der Grün-Wert und der Blau-Wert über 200 liegen, den Wert auf reines Weiß, also 255,255,255.

Filter können ganz verschiedene Aufgaben wahrnehmen, z.B.:

- Fehlerkorrekturen in Bildern durchführen (z.B. Farbkorrekturen)
- Interessante Effekte einbauen (z.B. Puzzle-Effekt oder Verwisch-Effekte)
- Optische Standardeffekte nachempfinden (z.B. Lichteinfall von einer Seite oder Mondscheinlicht)
- Erzeugung völlig neuer Phantasiegrafiken (dabei wird die Ausgangsgrafik nur als »zufälliger Input« benutzt)

Die folgenden Bilder zeigen die Anwendung einiger Filter auf eine Ausgangsgrafik.



24.21: Ausgangsgrafik





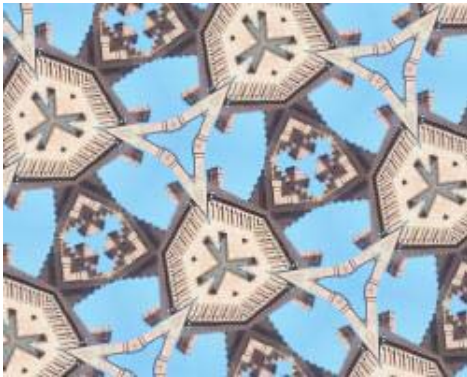
24.22: Filter, der einen Verzerrungseffekt bewirkt



24.23: Filter, der einen Kohlezeichnungs-Effekt bewirkt



24.24: Filter, der einen Spot-Effekt bewirkt



24.25: Filter, der die Grafik zum Erzeugen eines Musters benutzt

### 24.3.13 Anti-Aliasing

Anti-Aliasing ist ein Algorithmus für Pixelgrafiken, um sichtbare Treppeneffekte an harten Kanten oder Farbübergängen in Grafiken auszugleichen. An den folgenden Grafiken können Sie den Effekt gut beobachten.

|                                                                                     |                    |                                                                                     |                          |
|-------------------------------------------------------------------------------------|--------------------|-------------------------------------------------------------------------------------|--------------------------|
| Verdana                                                                             | ohne Anti-Aliasing |   | (Ausschnittvergrößerung) |
| Verdana                                                                             | mit Anti-Aliasing  |  | (Ausschnittvergrößerung) |
|  | ohne Anti-Aliasing |  | (Ausschnittvergrößerung) |
|  | mit Anti-Aliasing  |  | (Ausschnittvergrößerung) |

24.26: Anti-Aliasing

Die Ausschnittvergrößerungen zeigen das Prinzip des Anti-Aliasing. Mit Anti-Aliasing werden bei schrägen Strichen oder Rundungen die Treppeneffekte abgeschwächt, indem abgeschwächte Farbtöne hinzugefügt werden.

Besonders bei Schriftzügen innerhalb von Grafiken ist Anti-Aliasing ein Mittel für mehr Professionalität. Aber auch bei anderen Objekten mit Schrägen oder Kurven bietet sich Anti-Aliasing an, um die Ränder weich zu zeichnen und sichtbare Pixeltreppen zu vermeiden.

Gute Grafikprogramme unterstützen den Anti-Aliasing-Effekt. Das Problem dabei ist normalerweise, dass Pixelgrafiken keine »Objekte« enthalten, die man einfach markieren kann, um ihnen Eigenschaften – wie zum Beispiel den Anti-Aliasing-Effekt – zuzuweisen. Einige Programme

bieten jedoch die Möglichkeit an, Pixelbereiche, die ein erkennbar zusammenhängendes Muster darstellen, als Objekt zu markieren (Hilfsmittel dazu ist der meist so genannte »Zauberstab«). Markierten Objekten können Sie dann die Eigenschaft des Anti-Aliasing zuweisen. Beim Neuerstellen von Elementen wie Schriftzügen oder geometrischen Figuren können Sie dagegen von vorneherein den Anti-Aliasing-Effekt einstellen.

## 24.4 Animierte GIF-Grafiken

### 24.4.1 Allgemeines zu animierten GIF-Grafiken

Das GIF-Format unterstützt in der neueren 89er-Version die Möglichkeit, mehrere Grafiken in ein und derselben Grafikdatei zusammenzufassen. Die Einzelgrafiken der Datei werden nacheinander »abgespult« und angezeigt. Dabei ist es auch möglich, Kontrollbefehle mit abzuspeichern, z.B. Verzögerungszeiten zwischen den einzelnen Bildern. Dieses Feature bietet Ihnen zwei interessante Einsatzmöglichkeiten:

- Sie können eine automatisch ablaufende »Diashow« mehrerer verschiedener Bilder programmieren.
- Sie können eine Sequenz ähnlicher und aufeinander aufbauender Bilder in einer GIF-Datei speichern. Dadurch ergibt sich ein »Daumenkino-Effekt«, d.h. eine einfache Video-Animation.

Mit Hilfe von animierten GIF-Grafiken können Sie also bewegte Elemente auf Ihre Web-Seiten bringen, ohne in Programmiersprachen wie Java einsteigen zu müssen. Beachten Sie aber auch, dass animierte GIF-Grafiken schnell eine kritische Größe erreichen. Denn mehrere Einzelgrafiken in einer Grafikdatei benötigen zwangsläufig mehr Speicher. Animationen sollten Sie daher nur aus kleinen Grafiken erzeugen. Ferner ist im Hinblick aufs Web-Design darauf hinzuweisen, dass animierte Elemente unwillkürlich die Aufmerksamkeit des Betrachters auf sich lenken und dadurch aber von vielleicht wichtigeren Inhalten einer Seite, wie Navigation oder Information, ablenken. Setzen Sie animierte GIFs daher vorsichtig und mit Bedacht ein. Das sprichwörtliche Gezappel auf manchen Web-Seiten für überflüssiges Gimmicks ist mittlerweile verpönt und gilt als stümperhaft.

Das Erstellen animierter GIF-Grafiken wird von vielen professionellen Grafikprogrammen unterstützt. Daneben gibt es Spezialprogramme, mit deren Hilfe Sie bereits vorhandene Grafiken zu animierten Grafiken zusammenstellen können.

### 24.4.2 Diashow-Effekte durch animierte GIF-Grafiken

Diashow-Effekte können Sie z.B. verwenden, um in schneller Abfolge die typischen Produkte Ihrer Firma anzuzeigen oder um in einem interaktiven Lernprojekt zum Erlernen einer Fremdsprache neue Vokabeln kurz einzublenden (»unterbewusst lernen«). Sie können auch mehrere mit einer Gesamtbedeutung assoziierten Bilder in Reihe schalten und auf diese Weise animierte Symbole oder animierte Cliparts präsentieren. Wichtig ist, dass die Einzelbilder für den Anwender einen nachvollziehbaren Zusammenhang ergeben. Die Anzeigedauer der Einzelbilder können Sie frei einstellen.

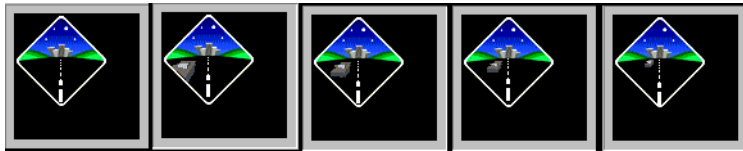


24.27: Typische Diashow-Effekte

### 24.4.3 Daumenkino-Effekte durch animierte GIF-Grafiken

Daumenkino-Effekte können sie z.B. verwenden, um Cliparts zu einfachen Trickfilmsequenzen zu erweitern, um wissenschaftliche Ablaufprozesse zu visualisieren oder um in der technischen

Dokumentation Handgriffe und andere zeitliche Abläufe zu verdeutlichen. Die Geschwindigkeit der Bildsequenzen und damit die Ablaufgeschwindigkeit können Sie frei einstellen.



24.28: Typische Daumenkino-Effekte

#### 24.4.4 Vorgehensweise beim Erstellen animierter GIF-Grafiken

Bevor Sie eine animierte Grafik erstellen können, müssen Sie folgende Vorbereitungen treffen:

- Sie müssen die Einzelbilder mit Hilfe eines geeigneten Grafikprogramms erstellen bzw. auswählen.
- Sie sollten alle Bilder in neue Grafikdateien mit einheitlicher Bildgröße und einheitlicher Farbpalette kopieren.
- Speichern Sie alle auf diese Weise erzeugten Bilder als GIF-Grafiken ab. Vergeben Sie dabei am besten durchnummerierte Dateinamen wie *bild1.gif*, *bild2.gif* usw., und zwar in der Reihenfolge, in der Sie die Bilder später als Einzelbilder in der animierten Grafik ablegen wollen.

Nachdem Sie diese Vorbereitungen getroffen haben, können Sie die Software zum Erstellen der animierten GIF-Grafik aufrufen. Da die Bedienung solcher Programme sehr unterschiedlich ist, wird hier nicht näher auf einzelne Bedienschritte eingegangen. In jedem Fall stehen Ihnen folgende Möglichkeiten zur Verfügung:

- **Bild (image):** Indem Sie eine GIF-Grafik einfügen, wird diese Grafik als Einzelbild in die animierte Grafik übernommen.
- **Schleife (loop):** Indem Sie eine Schleife einfügen, können Sie bestimmen, wie oft die gesamte Bildsequenz wiederholt werden soll. Sie können einen Zahlenwert editieren, der die Anzahl der Abspielwiederholungen angibt.
- **Kontrollelement (control):** Indem Sie ein Kontrollelement einfügen, können Sie den Ablauf zwischen zwei Einzelbildern kontrollieren. Sie können z.B. die Verzögerung bis zur Anzeige des nächsten Einzelbilds einstellen. Ferner können Sie Angaben zum transparenten Hintergrund der Einzelbilder machen.
- **Text (plain text):** Indem Sie Text einfügen, können Sie den Ablauf der animierten Grafik zusätzlich durch eingeblendete Textteile ergänzen. Dabei können Sie den Text formatieren.
- **Kommentare (comments):** Indem Sie Kommentare einfügen, können Sie Ihre Arbeiten intern auskommentieren. Solche Kommentare werden nicht mit angezeigt.

Um das fertige Ergebnis zu betrachten, brauchen Sie natürlich ein geeignetes Anzeigeprogramm. Einige Editierprogramme für animierte GIF-Grafiken bieten selbst die Möglichkeit des Betrachtens an. Wenn nicht, können Sie die Grafik auch mit Ihrem Web-Browser lokal öffnen und betrachten.

## 25 Web-Projektverwaltung

### 25.1 Web-Projekte planen

#### 25.1.1 Allgemeines zu Web-Projekten

Projekte im Web sollten stets zwei Anforderungen genügen: Sie sollten zum einen in sich geschlossen sein, d.h. dem Anwender im Web-Browser als ein erkennbar geschlossenes Ganzes erscheinen; zum anderen sollten sie aber auch der Tatsache Rechnung tragen, dass sie Teil einer weltweiten Plattform sind, in der es auch andere Projekte gibt, und vor allem die Möglichkeit zu thematisch sinnvollen Verweisen auf andere Projekte anbieten.

Nicht wenige Leute suchen mit ihren Homepages im Netz den schnellen Erfolg oder zumindest die tiefe Bewunderung der Seitenbesucher vor ihren Programmierheldentaten. Nachdem sie dabei in tagelanger Qual irgendwelches Zeug für Layout und Effekte zusammengeklaut haben, ohne sich um Inhalte zu kümmern, überfallen sie schließlich einschlägige Foren mit der Frage, wie sie ihren Quelltext vor den Augen der Besucher schützen können. Aber so funktioniert das Web nun mal nicht, und glücklicherweise bestraft sich dort jeder selbst, der nur profitieren will, ohne etwas zu geben.

Es gibt zwar kein Patentrezept für gelungene und erfolgreiche Web-Projekte, aber es gibt mittlerweile durchaus Erfahrungswerte und Empfehlungen für benutzerfreundliches und nutzbringendes Publizieren im Web. Der Erfolg muss auch nicht immer im direkten Geldfluss liegen. Ein, zwei interessante neue Kontakte, die sich im Lauf der Zeit über ein Web-Projekt ergeben, können viel mehr »einbringen« als mit Werbebannern zugekleisterte Seiten, deren Anbieter nur in Klickraten denkt. Und schließlich kann eine Homepage auch einfach nur Spaß machen.

Im Web präsent sein, heißt »in« sein. Aber belanglose Werbung oder Allerweltsinformation bringen weder dem Web noch Ihrem Geldbeutel etwas. Das Web braucht Juwelen, um eine Sensation zu bleiben. Und Juwelen sind teuer. Eigentlich können nur Besessene so viel aufbringen. Das Kapitel 25.1 fasst die Überlegungen zusammen, die Sie im Vorfeld eines Projekts anstellen sollten. Denn klare Konzepte und überlegtes Vorgehen sind schon mal der erste Schritt für ein gutes Projekt.

#### 25.1.2 Ziel des Projekts

Beispiele für mögliche Ziele eines Projekts:

- Wir als Firma stellen uns und unsere Produkte/Dienstleistungen vor (Kommunikation).
- Wir als Firma bieten eine Dienstleistung direkt über diese Web-Seiten an (Transaktion, Interaktion, Partizipation).
- Ich als Einzelperson stelle meine Aktivitäten in meinem Fachgebiet/Hobby vor.

- Ich als Einzelperson veranstalte auf diesen Web-Seiten ein persönliches Happening.
- Wir als Institution/Organisation/Verband/Verein stellen unsere Ziele und Aktivitäten vor.
- Wir als Institution/Organisation/Verband/Verein bieten Ihnen die Möglichkeit, über diese Web-Seiten an einer bestimmten Aktion teilzunehmen.
- Ich/Wir stelle(n) auf diesen Web-Seiten einen großen Datenbestand aus einem bestimmten Sachgebiet zur allgemeinen Verfügung.
- Ich/Wir sammeln auf diesen Web-Seiten in loser Weise alles, was zu einem bestimmten Sachgebiet gehört.

Diese Liste mit Beispielen will zeigen, dass es erstens sehr unterschiedliche Ziele eines Web-Auftritts geben kann, und dass auch ein und derselbe Anbieter von Web-Seiten unterschiedliche Ziele verfolgen kann. Wichtig ist, dass Sie sich in der Konzeptphase eines Web-Projekts über das oder die Ziele des Projekts im Klaren sind, und dass Sie dem Anwender, der Ihre Seiten aufruft, gleich zu Beginn (auf der Einstiegsseite) klar machen, welches Ihre Ziele sind. Das kann explizit geschehen durch Sätze wie: »XY begrüßt Sie. Auf diesen Seiten stellen wir uns und unsere innovative Technik vor usw.«, oder »Light your fire and feel free to loose yourself into this hyper-space«. Es kann aber auch durch die Aufmachung der Startseite geschehen. So vermittelt eine neutral aufgebaute Startseite (ohne Hintergrundbilder, animierte Grafiken, JavaScript usw.) wissenschaftliche Seriosität; eine gestylte, aber überlegen statische Seite suggeriert beim Anwender häufig eine große, etablierte Firma oder Institution oder ein Projekt von monumentalen Ausmaßen. Eine poppige, nach Aufmerksamkeit schreiende Seite dagegen verrät eine kleine, werbende Firma oder ein freakiges, nicht ganz so ernst zu nehmendes Projekt.

### 25.1.3 Zielgruppe des Projekts

Genauso wichtig wie eine Zielsetzung ist es, die Zielgruppe zu bestimmen, die durch das Projekt angesprochen werden soll. Mögliche Zielgruppen sind z.B.:

- Potenzielle Kunden/Käufer
- Hobbyisten/Liebhaberkreise
- Wissenschaftler/Experten
- Kinder und Jugendliche
- Familien
- Computerfreaks/Hacker
- Vergnügungssüchtige
- Neugierige
- Sinnsuchende

Natürlich können Sie versuchen, alle Besucher Ihrer Web-Seiten zufrieden zu stellen. Das wird allerdings ziemlich schwer, und es besteht die Gefahr, dass Sie niemanden so richtig zufrieden stellen. Besser ist es, sich vorzustellen, wie bestimmte Leute, die auf Ihre Web-Seiten stoßen, sofort ein Lesezeichen darauf setzen, weil sie gleich erkennen, dass dies Seiten sind, die mit ihren

eigenen Interessen, ihrem Wissensdurst, ihrer Lebensart oder ihren Wünschen, Gedanken und Hoffnungen zu tun haben. Es gibt beispielsweise Web-Seiten, auf denen Betroffene über eine nicht alltägliche Krankheit berichten, unter der sie leiden. Wer nun selbst von dieser Krankheit betroffen ist, im Web nach Informationen darüber sucht und auf diese Seiten stößt, wird sie verschlingen, und sie werden ihm viel bedeuten. Solche Seiten haben eine ziemlich eingeschränkte Zielgruppe, aber dafür sind sie für ihre Zielgruppe Gold wert.

Ebenso wie die Zielvorgabe Ihres Projekts sollte dem Anwender auch die adressierte Zielgruppe klargemacht werden. Dazu eignen sich explizite Sätze wie: *»Allen Fahrradsüchtigen bieten wir auf diesen Seiten technische Neuigkeiten und Tourenvorschläge«*. Aber auch die Aufmachung der Startseite selektiert das Publikum. Dabei hängt allerdings viel vom Zusammenspiel der Elemente ab. Ein dunkler Sternenhimmel als Hintergrundbild spricht joystickverliebte Cybernauten ebenso an wie entrückte Esoteriker. Erst die Vordergrundsymbolik (z.B. grelle Strahlen aus Laserwaffen, die ein Monster treffen, oder wehender weißer Schleier, der auf eine Pyramide zufliegt) sorgt in solchen Fällen für intuitiv klare Verhältnisse.

#### 25.1.4 Projektumgebung im Web erforschen

Vor allem, wenn Sie im Web etwas verkaufen oder mit einer Idee erfolgreich sein wollen, sollten Sie bereits in der ersten Planungsphase herausfinden, ob es zu dieser Idee schon etwas gibt im Web, und wenn ja, was und mit welchem Erfolg. Überlegen Sie sich zu diesem Zweck ein paar einschlägige Stichwörter, die Ihr Projekt charakterisieren. Suchen Sie nach diesen Stichwörtern in großen Suchmaschinen.

Werten Sie die Suchergebnisse nach folgenden Kriterien aus:

- **Gibt es gar nichts zu diesen Suchbegriffen?** Das kann bedeuten, dass die Welt auf genau das wartet, was Sie anbieten wollen. Es kann aber auch bedeuten, dass das niemanden in der Welt interessiert. In diesem Fall ist es angebracht, mit einer kleinen, nicht so aufwendigen Projektversion zu beginnen und Feedback abzuwarten.
- **Gibt es uferlos viel zu den Suchbegriffen?** Das kann bedeuten, dass Ihre Pläne voll im Trend liegen, aber auch, dass Sie hier nicht mehr viel Neues werden bieten können. Folgen Sie den Verweisen der Suchergebnisse und analysieren Sie sorgfältig, was es bereits gibt und wie Sie sich von bestehenden Angeboten nachvollziehbar abgrenzen können (z.B. *»erstes Angebot dieser Art im südpfälzischen Raum«*).

#### 25.1.5 Inhalte und Redaktion

Anfänger unterliegen häufig dem Irrglauben, ein »geiles Layout« oder ein paar tolle Gimmicks würden ein Web-Projekt interessant machen. Die Erfahrung zeigt das Gegenteil: Anwender besuchen neue, fremde Web-Projekte normalerweise nicht, weil sie ganz besoffen sind aus lauter Vorfreude auf viel buntes Zeug am Bildschirm, sondern weil sie etwas ganz Bestimmtes suchen und das aufgesuchte Projekt als aussichtsreichen Kandidaten für ihr Anliegen auserkoren haben. Solche Besucher erwarten zwar durchaus ein ansprechendes Layout, aber in erster Linie wollen sie möglichst schnell ihr Ziel erreichen. Das Ziel aber ist nicht das »Betrachten« und »Verweilen« auf Ihren Seiten, sondern meistens ein inhaltliches. Die Besucher wollen Informationen. Auch wenn es ein elektronischer Shop ist, wollen die meisten Besucher mehr über Produkte erfahren



als nur Artikelnamen und Preise. Und selbst wenn es einfach eine private Homepage ist, wollen die Besucher etwas über Sie, Ihre Gedanken usw. erfahren. Das Web ist in erster Linie ein Informationsvermittelndes Medium, und dem sollten Web-Projekte Rechnung tragen.

Doch Inhalte zu erstellen ist aufwändig, Inhalte zu pflegen und zu aktualisieren nicht minder. Dazu ist redaktionelle Arbeit erforderlich. Viele, die im Web publizieren, sind zuvor noch nie oder nur am Rande mit redaktionellen Tätigkeiten in Berührung gekommen. Der anfängliche Leitspruch des W3-Konsortiums »Everyone's a publisher!« beinhaltet eben nicht nur die verlockenden neuen technischen Möglichkeiten, ein Publikum zu erreichen, sondern auch die ganze Arbeit eines »Publishers«, die damit verbunden ist. Die Textarbeit wird viel zu häufig gegenüber dem Programmieraufwand eines Web-Projekts unterschätzt. Und genau aus diesem Grund scheitern viele sehr gut programmierte Web-Projekte.

Im Planungsstadium eines Web-Projekts müssen Sie sich deshalb unbedingt Gedanken machen über folgende Punkte:

- **Startinhalte:** Woher sollen die ersten Inhalte kommen? Gibt es vorhandene, geeignete Texte? Gibt es beispielsweise ein Firmenprospekt, wissenschaftliche Aufsätze, Dokumentarchive, Datenbanken? Sind diese Quellen wirklich geeignet – enthalten sie beispielsweise keine vertraulichen, nicht für die Öffentlichkeit geeigneten Daten? Müssen die Quellen also noch ausgewertet werden? Und wenn gar keine Quellen vorhanden sind – wer erstellt dann die Inhalte? Sind dazu Recherchen erforderlich? Wer liest Korrektur?
- **Laufende neue Inhalte:** Woher sollen laufende neue Inhalte kommen? Besucher von Web-Projekten werden nur dann zu Stammesbesuchern, wenn es regelmäßig etwas Neues auf den Seiten zu entdecken gibt. Sorgen Redakteure oder Zulieferer für neue Inhalte? Oder sollen neue Inhalte von den Besuchern selbst beigetragen werden, beispielsweise durch Anbieten eines Diskussionsforums?
- **Aktualität:** Wer überwacht die Aktualität und Richtigkeit der Inhalte? Wer ändert im Web-Datenbestand geänderte Fakten, z.B. Informationen zu Produkten, die vom Markt genommen wurden, »Nachrichten von gestern« usw.? Wer repariert nicht mehr funktionierende Verweise zu anderen Projektseiten oder zu nicht mehr existierenden anderen Web-Adressen?

### 25.1.6 Kostenaufwand

Der Kostenaufwand eines Web-Projekts verteilt sich auf folgende Bereiche:

- Konzept- und Planung
- Realisierung
- Pflege

In allen Fällen handelt es sich um Zeitaufwand, der nicht zu unterschätzen ist. Wenn Sie aus Begeisterung oder als Hobbyist arbeiten, werden Sie die Formel  $\text{Zeit} = \text{Geld}$  nicht so sehr empfinden, aber wenn Sie kommerziell arbeiten, müssen Sie für alle drei genannten Bereiche Kalkulationen anstellen.

Wenn Sie Geld für das Projekt investieren müssen, sind folgende Abwägungen wichtig:

- **Selbst- oder Fremdarbeit:** Ist es günstiger, sich selbst die Zeit (Geld) zu nehmen, um sich in Konzept und Realisierung (fachlich/technisches Know-how) einzuarbeiten, oder ist es besser, jemanden zu beauftragen (ebenfalls Geld), der sich damit auskennt und dies übernimmt?
- **Rentabilität:** Wie empfindlich treffen Sie die anfallenden Kosten? Können Sie sich leisten, dass das Projekt ein Flop wird? Wenn ein Flop Ihren Untergang bedeutet, sollten Sie es vielleicht erst mal mit risikoloserer Investitionen versuchen.

Bei kommerziellen Web-Projekten ist es mittlerweile meist so, dass die erste Erstellung als Auftragsarbeit an Web-Agenturen vergeben wird. Die laufende Pflege der Inhalte einer solchen Web-Agentur zu überlassen hat sich dagegen als unrentabel erwiesen. Das bedeutet: Bei der Auftragsvergabe müssen bereits Regelungen getroffen werden, wie Sie als Auftraggeber anschließend das Projekt inhaltlich pflegen können. Immer häufiger verlangen Auftraggeber von den Web-Agenturen deshalb die Implementierung eines so genannten **Content Management Systems (CMS)**. Ein solches System erlaubt webtechnisch unerfahrenen, aber inhaltlich kompetenten und redaktionell verantwortlichen Mitarbeitern des Auftraggebers, über eine anwenderfreundliche Internet-Schnittstelle (häufig webbasierte Lösungen) neue Inhalte in das Projekt einzuarbeiten. Es gibt fertige Softwarelösungen für CMS, aber oft werden auch individuelle Lösungen programmiert. Professionelle CMS-Software beinhaltet gegenüber einfachen Lösungen Features wie »Redaktionsvorstufe« und »endgültiges Freischalten durch die Redaktionsleitung« usw.

Im Linkverzeichnis des Online-Angebots von SELFHTML unter *selfaktuell.teamone.de* finden Sie Verweise zum Thema »Content Management Systeme«.

### 25.1.7 Terminierung und Ankündigung

Egal, ob Sie ein privates oder ein professionelles Web-Projekt planen – eines sollten Sie in jedem Fall tun: sich Termine setzen. Denn ohne selbst gesetzte Termine wird ein Projekt nie fertig. Die Termine sollten allerdings realistisch sein. Ein Termin, der nicht zu halten ist, ist nicht mehr wert als kein Termin. Für einfache Projekte reicht eine schlichte »Deadline«. Bei komplexeren Projekten, an deren Realisierung mehrere Hände oder unabhängige Parteien arbeiten, ist möglicherweise Terminkoordinierung mit Hilfe von Projektplanungssoftware erforderlich, ähnlich wie bei Produktrealisierungen oder Bauprojekten.

Gleichzeitig stellt sich die Frage nach einer Vorankündigung. Häufig wird ein Domain-Name reserviert, sobald eine Projektidee geboren ist. Dagegen ist auch nichts einzuwenden. Aber nach kurzer Weile steht dann der angemietete Speicherplatz im Web zur Verfügung, und die Adresse ist öffentlich aufrufbar. In dieser Phase neigen viele Heißsporne dazu, irgendwelche prahlerischen Vorankündigungen zu veröffentlichen, mit lustigen animierten Baustellenschildern oder moderneren Flash-Trailern versehen, und das Ganze wird dann sogleich in allen bekannten Suchmaschinen angemeldet.

Vorankündigungen können unter bestimmten Umständen sinnvoll sein, so etwa, um auf eine neu reservierte Marke schon mal vorab hinzuweisen. In den meisten Fällen aber sind Vorankündigungen ein Ärgernis. Die Suchmaschinen werden auf solchen Seiten mit unnützem Müll gefüttert, und Besucher, die den Anbieter nicht kennen, werden sich kaum von dessen großspurigen Ankündigungen beeindrucken lassen. Stattdessen klicken sie genervt weg, sobald sie begriffen haben, dass da nur wieder einer dieser Prahlschneisen am Werk ist.

Vorankündigungen sollten so sein wie das geplante Web-Projekt: fundiert, informationsorientiert und benutzerfreundlich. Also keine sinnlosen Animationen, die Ladezeit verursachen, sondern stattdessen ein paar oder mehr Sätze über das geplante Projekt.

### 25.1.8 Gliederung und Strukturierung

Ein Web-Projekt besteht aus wenigstens einigen, oft aus vielen, manchmal aus unzähligen Web-Seiten. Je mehr Seiten und zugehörige Dateien das Projekt enthält, desto wichtiger wird eine schriftliche Fixierung der Datei- bzw. Verweisstruktur. Aber auch bei kleinen Projekten, die nur aus wenigen Seiten bestehen, sollten Sie sich vorher überlegen, was auf welche Seite gehört und wie die Seiten vernetzt werden sollen.

Denken Sie stets daran, dass das Web ein Hypertext-Medium ist. Hypertext zwingt Sie nicht zu bestimmten Formen. Hypertext ist ein nichtlineares Medium. Der Anwender soll nicht der Reihe nach lesen, sondern auswählen, was ihn interessiert. Das ist eine große Freiheit, birgt jedoch zugleich die Gefahr des »lost in hyperspace«-Gefühls.

Die meisten Anwender im Web sind keine erfahrenen Hypertext-Piloten. Bei Projekten, die aus mehr als einer Handvoll Seiten bestehen, sollten Sie deshalb versuchen, dem Anwender eine »Metapher« zu bieten, an die er sich halten kann. Mögliche Metaphern:

- **Buchmetapher:** Das Projekt hat Dokumentcharakter, besitzt ein Inhaltsverzeichnis und evtl. ein Stichwortverzeichnis, Dateien entsprechen Kapiteln usw. Diese Metapher eignet sich besonders für begrenzte Information zu bestimmten Themen, wobei die Daten selten aktualisiert werden.
- **Verzeichnis- oder Baummetapher:** Das Projekt wirkt auf den Anwender wie ein Dateimanager. Es gibt Dateien mit Verweislisten, die durch ein Ordnersymbol gekennzeichnet werden, und Dateien mit Information, die durch Dokumentsymbole gekennzeichnet werden. Diese Metapher eignet sich besonders für große Datenbestände mit häufigen Neuzugängen, wobei jedoch der Informationsbereich, der abgedeckt wird, von vorneherein bekannt ist.
- **Indexmetapher:** Ein großes, alphabetisch oder nach anderen Kriterien (z.B. Datum, Zeit) sortiertes Verweisverzeichnis bildet den Zugang zur Information. Diese Metapher eignet sich besonders für Projekte, deren Informationsbereich nicht genau eingrenzbar ist und bei denen häufig neue Daten hinzukommen.

Sobald der Anwender Ihre beabsichtigte Metapher erkennt, kann er sich eine »räumliche Vorstellung« Ihres Projekts machen. Das ist eine ganz wichtige Sache, die Sympathie des Anwenders gegenüber dem Projekt hängt tatsächlich zum großen Teil davon ab, ob er diesen Aha-Effekt hat oder nicht.

### 25.1.9 Corporate Design

Genauso wichtig wie ein Gliederungskonzept ist ein Corporate-Design-Konzept. Das bedeutet, dass alle Dateien, die zu einem Projekt gehören, ein optisch einheitliches Bild bieten, damit der Anwender intuitiv erkennt, dass er sich noch im gleichen Projekt befindet. Corporate Design stellen Sie her, indem Sie bestimmte Gestaltungselemente bei jeder Datei des Projekts wieder verwenden. Das können sein:

- **Logo:** Es sollte an immer gleicher Stelle auf der Seite stehen, zum Beispiel immer links oben am Seitenanfang. Wenn Sie mit der Frame-Technik arbeiten, können Sie das Logo auch in einem fixen Bildschirmbereich platzieren.
- **Grafische Symbole:** Das sind kleine Grafiken, die an bestimmten Stellen stehen und eine für den Anwender sofort erkennbare bzw. nachvollziehbare Bedeutung haben. Symbole können z.B. bedeuten: »dies ist ein Beispiel«, oder »dies ist eines unserer Produkte« oder »dies ist ein Verweis, mit dem der Anwender das Projekt verlässt«.
- **Textformate:** Dazu gehört z.B., Beispiele immer in diktengleicher Schrift darzustellen.

Corporate Design suggeriert Geschlossenheit. Besonders bei kommerziellen Projekten, also bei Firmen, die sich vorstellen oder direkt etwas anbieten, vermittelt Corporate Design dem Anwender das Gefühl, es mit erfahrenen Profis zu tun zu haben.

Corporate Design ist etwas anderes als Web-Design. Aber Web-Design muss die Faktoren, die das Corporate Design vorgibt, im Design von Layout, Navigation, Textformatierung usw. berücksichtigen. Gerade wenn Sie eine Fremdbeauftragung anstreben, ist es wichtig, dem Auftragnehmer genaue Vorgaben zum Corporate Design zu machen. Hören Sie aber andererseits auch auf mögliche Einwände. Corporate Design darf nicht die Ergonomie beim »Anwenden« der Seiten beeinträchtigen.

### 25.1.10 Web-Design

Über gutes und schlechtes, richtiges und falsches Web-Design wird in Fachkreisen endlos diskutiert. Die folgenden Regeln fassen wichtige Punkte zusammen:

- **Form ohne Inhalt ist leer:** Ohne vermittelnde Information, ohne echte Unterhaltung kann eine Web-Seite optisch noch so perfekt sein – sie ist dennoch eine Zumutung, und niemand, der sie besucht hat, kommt ein zweites mal wieder. Wer nichts zu bieten hat außer ein paar schicken Bildschirmfüllungen, wird im Web keine dauerhaften Freunde gewinnen.
- **Inhalt ohne Form ist langweilig:** Ohne sinnvolle Strukturierung in kleine Informationseinheiten und ohne Auflockerung durch Grafiken oder multimediale Elemente mit Signal-, Symbol- oder Illustrationswirkung ist Informationsaufnahme am Bildschirm ermüdend und nervtötend. Bildschirmseiten lassen sich nicht mit Buchseiten vergleichen.
- **Navigation ist angesagt:** Ein Web-Projekt mit viel Informationsgehalt ist zunächst einmal ein Hypertext-Projekt, und der Eindruck, den es beim Anwender hinterlässt, hat viel damit zu tun, ob es einfach ist, darin die gewünschten Informationen zu finden, ob es Spaß macht, seine eigenen Spuren darin zu verfolgen, und ob es leicht ist, jederzeit und schnell zu anderen wichtigen Stellen des Projekts zu verzweigen. Web-Design ist also mindestens ebenso viel Hypertext-Design wie Grafik-Design.
- **Plattformunabhängigkeit ist angesagt:** Während ein herkömmlicher Designer vorher weiß, ob sein Entwurf auf einem Bierdeckel oder auf einer Plakatwand landet, kann der Web-Designer nicht wissen, auf welchen Anzeigegeräten seine Web-Seiten letztendlich ihre Wirkung entfalten sollen. In den meisten Fällen sind unproblematische Designs besser, die mit klaren, einfachen Formen aufwarten, gute farbliche Kontraste besitzen und keine üppigen Breiten und Höhen erzwingen. Auch an so genannte Benutzerrandgruppen wie Sehbehinderte sollte gedacht werden, in welcher Form auch immer.

- **Das Web gehört zum Internet:** Eine Web-Seite hat nicht nur möglichst gut auszusehen, sie sollte auch möglichst wenig Bytes haben. Dass die Bandbreiten für die Datenübertragung beim Durchschnittsanwender allmählich besser werden, bedeutet nicht, hier und jetzt zu Klotzen, ohne Rücksicht auf den Netzverkehr zu nehmen. Auch andere Dinge sind zu berücksichtigen. Gute, aussagekräftige Stichworte und Kurzbeschreibungen in den Meta-Angaben sind ebenso wichtig für den Erfolg einer Web-Seite wie ihre optische Wirkung. Denn eine Web-Seite wird nicht nur von Menschen besucht, sondern auch von Programmen, die sie für Suchmaschinen im Internet indizieren.

## 25.2 Web-Projekte publizieren

### 25.2.1 Allgemeines zum Publizieren von Web-Projekten

Um eigene Projekte im WWW zu publizieren, brauchen Sie:

- **Einen Internet-Zugang:** Wenn Sie noch keinen Internet-Zugang haben, müssen Sie sich einen Anbieter (Provider) suchen, der Ihnen einen Zugang vermittelt. Bei der Auswahl eines geeigneten Zugangs-Providers spielen verschiedene Faktoren eine Rolle, zum Beispiel die Tarife der Provider, Sonderkonditionen, Volumen- oder Zeitbeschränkungen usw. Nachdem Sie sich bei einem Provider angemeldet haben (bei Call-by-Call-Providern ist mittlerweile auch keine Anmeldung mehr erforderlich), erhalten Sie eine User-ID und ein Passwort. Mit diesen Daten können Sie dann die Internet-Verbindung aufbauen, sei es über Telefonleitung, TV-Kabel, Stromnetz, Satellit, Handy oder eine andere Trägertechnik. Voraussetzung ist allerdings, dass Ihr Rechner internetfähig ist. Dazu gehört eine so genannte Internet-Socket-Software (unter Windows z.B. die Datei *winsock.dll*) sowie geeignete Internet-Client-Programme (etwa ein Web-Browser und ein E-Mail-Programm).
- **Festspeicherplatz auf einem Server-Rechner:** Das sollte ein Rechner sein, der eine permanente und sehr schnelle Internet-Anbindung hat, mindestens die Internet-Protokolle HTTP und FTP unterstützt und von seiner Hard- und Software-Grundausstattung so ausgelegt ist, dass er viele Besucher gleichzeitig verkraftet. In der Regel werden Sie den entsprechenden Speicherplatz im Web bei einem Hosting-Provider anmieten, der über entsprechende Technik verfügt. Es gibt eine Fülle von Anbietern, die Speicherplatz im Web vermieten. Die Leistungen und Preise sind sehr unterschiedlich. Sie müssen herausfinden, welche Leistungen Sie brauchen und wo Sie diese Leistungen am günstigsten bekommen. In vielen Fällen können Sie bei dem Provider, bei dem Sie Ihren Internet-Zugang haben, auch Speicherplatz für eigene Web-Projekte mieten.
- **Eine Web-Adresse:** Dabei gibt es unterschiedliche Typen. Bei Online-Diensten oder Billig-Angeboten erhalten Sie beispielsweise häufig Adressen vom Typ *http://www.IhrProvider.com/IhrVerzeichnis/*. Den Verzeichnisnamen dürfen Sie sich bei der Anmeldung normalerweise selbst aussuchen. Wenn Sie eine eigene Domain-Adresse wollen, also beispielsweise eine Web-Adresse vom Typ *http://www.IhrName.com/*, muss diese bei einer Vergabebehörde im Internet beantragt werden. Provider, über die Sie Speicherplatz mieten können, übernehmen die entsprechende Anmeldung für Sie.

- **Einen Zugang zu dem gemieteten Speicherplatz:** Dazu erhalten Sie von dem Provider, bei dem Sie den Speicherplatz mieten, eine Kennung. Die besteht normalerweise aus einer User-ID und einem Passwort. In den meisten Fällen können Sie Ihr Passwort jederzeit selbst ändern.
- **Software zum Hochladen und Verwalten der eigenen Daten auf dem Server-Rechner:** Der klassische Internet-Dienst zum Übertragen von Dateien und zum Ausführen einfacher Dateioperationen auf dem Server-Rechner (Löschen, Kopieren, Verschieben, Zugriffsschutz festlegen usw.) ist das File Transfer Protocol (FTP). Dazu gibt es FTP-Programme, die Sie auf Ihrem Rechner installieren müssen. Bei manchen Anbietern können oder müssen Sie spezielle Software verwenden, die Ihnen der Provider zur Verfügung stellt.

Speicherplatz im Web, der so genannte Webspace, will mit Bedacht ausgewählt sein. Welches Angebot Sie wählen sollten, hängt einerseits von der Art und den technischen Ansprüchen des Projekts ab, andererseits von den Kosten, die Sie bereit zu tragen sind. Generell gilt: Sparen Sie nicht am falschen Ende! Viele Leute meinen, sie könnten kostenlos Speicherplatz bekommen und dabei all die tollen Sachen haben, die sie woanders sehen. Dem ist aber nicht so. Für den Preis, den das Monats-Abo einer Tageszeitschrift kostet, bekommen Sie bereits ordentliche Angebote, die für die Mehrzahl der Web-Projekte ausreicht. Doch leider gibt es auf dem Provider-Markt immer noch manche Stümper, die zwar erfolgreich mit Versprechungen locken, aber technisch völlig inkompetent sind. Vor allem am Support hapert es bei vielen Providern. Suchen und vergleichen Sie also lieber ein wenig länger, fragen Sie nach Erfahrungen anderer Projekt-Betreiber und entscheiden Sie sich nicht unbedingt für das erstbeste Angebot, das mit den auffälligsten Schriftgrößen und den kernigsten Sprüchen wirbt.

Im Linkverzeichnis des Online-Angebots von SELFHTML bei [selfaktuell.teamone.de](http://selfaktuell.teamone.de) aktuell finden Sie Verweise zu Web-Seiten, auf denen Provider-Angebote aller hier vorgestellten Typen aufgelistet und verglichen werden.

### 25.2.2 Homepages bei Online-Diensten

Diese Lösung können Sie anvisieren, wenn Sie bereits einen Internet-Zugang über einen Online-Dienst (T-Online, CompuServe, AOL usw.) haben und einfach nur eine private oder interessensorientierte Homepage im Internet publizieren wollen. Der Vorteil ist dabei, dass Sie keine Zusatzkosten haben, weil das Angebot in den Kosten für den Internet-Zugang bereits enthalten ist.

Bei den genannten Online-Diensten erhalten Sie in der Regel mehrere Megabyte Speicherplatz auf einem Server-Rechner des Online-Dienstes. Dorthin können Sie HTML-Dateien, Grafiken und meistens auch Multimedia-Dateien hochladen, die dann als Web-Seiten öffentlich zugänglich sind. Es gibt jedoch auch Beschränkungen. So bieten die Online-Dienste in der Regel nicht die Möglichkeit einer eigenen CGI-Schnittstelle an, wo Sie eigene Scripts ablegen können. Wenn Sie solche Dinge wie Zugriffszähler, Gästebücher oder Nachrichtenforen nutzen wollen, sind Sie auf öffentliche CGI-Services angewiesen. Auch weiterführende Zusatzleistungen wie Statistik, die Möglichkeit geschützter Verzeichnisse usw., sind in diesen Angeboten normalerweise nicht enthalten.

Wenn Sie bereits einen Zugang zu einem Online-Dienst haben und erwägen, über diesen Anbieter Ihre Homepage zu publizieren, erkundigen Sie sich bitte innerhalb des Online-Dienstes

danach, was Sie brauchen und wie alles im Einzelnen funktioniert. Bei CompuServe, AOL oder T-Online gibt es entsprechende Foren, in denen Sie diese Fragen stellen können.

### 25.2.3 Kostenlose Homepages

Diese Lösung ist sinnvoll, wenn Sie bereits einen Internet-Zugang haben, Ihr Zugangs-Provider aber keine Möglichkeit anbietet, Ihre eigene Homepage ins Web zu stellen. Ernsthaft in Erwägung ziehen sollten Sie diese Möglichkeit jedoch nur für private Homepages ohne Anspruch sowie für reine Spaßprojekte. Denn bei fast allen Anbietern von kostenlosem Webspace wird beim Aufruf der Web-Seiten automatisch Werbung eingeblendet. Die meisten Besucher der Seiten empfinden diese Werbung als lästig und halten sich kaum freiwillig länger auf solchen Seiten auf. Dagegen können Sie auch nichts weiter unternehmen, da die Web-Seiten, bevor sie zum Anwender in den Browser gelangen, durch ein Script auf dem Server-Rechner gelotst werden, das die automatisch aufpoppende oder in ein Frameset integrierte Werbung »einbaut«.

Bei den kostenlosen Anbietern erhalten Sie in der Regel mehrere Megabyte Speicherplatz, was für die meisten Privatprojekte durchaus ausreicht. Die Beschränkungen sind ähnlich wie bei den Online-Diensten. Eine Übersicht solcher Anbieter finden Sie beispielsweise bei [www.kostenlos.de](http://www.kostenlos.de).

### 25.2.4 Eigene Domain-Adresse

Wenn Sie eine eigene Domain-Adresse für Ihr Projekt haben möchten, müssen Sie sich einen geeigneten Namen ausdenken. Bei den Domain-Namen der »Top-Level«-Typen `.de` (deutsche Adressen), `.at` (österreichische Adressen), `.ch` (schweizer Adressen), `.com` (kommerzielle Adressen), `.net` (internetbezogene Adressen) oder `.org` (Organisationen) brauchen Sie mittlerweile viel Phantasie, denn fast alle einfachen Namen und Wörter sind längst vergeben. Beliebte sind daher zusammengesetzte Namen mit Bindestrichen geworden, wie beispielsweise `heute-geh-ich-ins-kino.de`. Oder Sie weichen auf die neuen Top-Level-Typen wie `.info` oder `.name` aus. Fragen Sie dazu gegebenenfalls den Provider Ihrer Wahl.

Der Provider übernimmt in den meisten Fällen die Anmeldung Ihres Domain-Namens. Häufig bieten Provider auch die Möglichkeit an, Sub-Domains einzurichten. Wenn Sie beispielsweise eine Domain namens `moecht-ich-gern.de` haben, dann können Sie Subdomains wie `das.moecht-ich-gern.de` oder `liebe.moecht-ich-gern.de` einrichten lassen. Die entsprechenden Web-Seiten wären dann beispielsweise aufrufbar unter <http://das.moecht-ich-gern.de/>.

Gerade bei Subdomain-Namen kann man mit etwas Phantasie mehrere attraktive, kurze und gut zu merkende Namensadressen haben. Subdomains sind für Provider technisch problemlos einrichtbar und erfordern keine weiteren Anmeldungen. Fragen Sie den Provider Ihrer Wahl nach Sub-Domains!

### 25.2.5 Virtual Hosting mit eigener Domain

Beim Virtual Hosting mieten Sie eine bestimmte Menge Speicherplatz. Den Rechner, der normalerweise beim Provider steht und auf den Sie dann Ihre Web-Seiten hochladen können, teilen Sie sich dabei normalerweise mit anderen Kunden des Providers, die ebenfalls ein solches Angebot gewählt haben. »Virtual« heißt diese Form deshalb, weil es bei solchen Web-Angeboten kein

1:1-Verhältnis zwischen der IP-Adresse des Rechners und dem Domain-Namen gibt, eben weil meistens mehrere Web-Angebote, alle mit eigenen Domain-Namen, auf dem gleichen Rechner liegen.

Virtual Hosting ist für einfache bis umfangreichere, werbefreie Homepages mit eigener Domain-Adresse, also elektronische Visitenkarten und Firmenportraits, aber auch seriöse Fachpublikationen oder Feature-Seiten zu einem bestimmten Thema meistens ausreichend und gut geeignet.

Bei Angeboten dieser Art erhalten Sie neben dem Speicherplatz in der Regel auch ein eigenes CGI-Verzeichnis, um serverseitig dynamische Web-Seiten zu erzeugen, sowie Zusatzleistungen wie Statistiken über Besucher Ihrer Seiten, oder einen Service-Bereich, in dem Sie – meist web-basiert und ohne zusätzliche Kenntnisse – Ihre persönlichen Kenndaten ändern können, Verzeichnisse schützen können, Online-Hilfen erhalten usw. Außerdem beinhalten diese Angebote in der Regel auch den Anspruch auf Support, wenn es mal technische Probleme gibt (Ihre HTML- und JavaScript-Probleme löst ein solcher Support allerdings nicht). Zusatzleistungen dieser Art haben Sie bei Online-Diensten und kostenlosen Webspace-Angeboten normalerweise nicht.

### 25.2.6 Server-Hosting mit eigener Domain

Beim Server-Hosting stellt Ihnen der Provider einen kompletten Rechner zur Verfügung, auf dem kein anderes Web-Angebot abgelegt ist. Beim Server-Hosting erhalten Sie in der Regel auch den vollen Betriebssystemzugang zum Server-Rechner, können selbst Benutzer einrichten, neue Software installieren usw. Über Internet-Dienste wie Telnet oder SSH können Sie sich dann auf dem Rechner einloggen und dort arbeiten. Dazu benötigen Sie entsprechende Telnet- oder SSH-Programme. Um sinnvoll auf einem Server-Rechner arbeiten zu können, sind gute Kenntnisse des dort eingesetzten Betriebssystems erforderlich. Wenn der Server-Rechner beispielsweise ein Linux-Betriebssystem hat, müssen Sie sich mit Linux gut auskennen, um die Möglichkeiten zu nutzen, die Ihnen ein eigener Server-Rechner bietet.

Server-Hosting ist für Web-Angebote erforderlich, die sehr umfangreich sind und/oder sehr hohe Besucherfrequenzen haben und/oder technisch nicht triviale Mittel einsetzen, für die beispielsweise spezielle Software auf dem Server erforderlich ist. Das können spezielle Server-Typen sein, etwa für Chats oder Multimedia, oder auch spezielle Datenbanksoftware.

Da bei solchen Web-Projekten meistens mehrere Personen Zugang und Kompetenzen haben, muss vieles organisiert werden. So sollte es einen »Webmaster« geben, also einen Betreuer, der den Server-Rechner, seine Software, seine gespeicherten Inhalte, Konfigurationen usw. kennt und beurteilen kann, ob und welche Hard-/Softwareerweiterungen angesichts der Server-Belastung nötig sind. Konfigurationsdaten und Konzepte zur Datenstruktur usw. sollten ausreichend dokumentiert werden.

Beim Server-Hosting sollten Sie noch stärker als beim Virtual Hosting auf ordentlichen Support achten. Es gibt Provider, die zwar für einen guten Preis eigene Server-Rechner anbieten, aber nicht zur Stelle sind, wenn es brennt.



### 25.2.7 Server-Housing mit eigener Domain

Beim Server-Housing stellen Sie selbst den Server-Rechner zur Verfügung, und der Provider stellt ihn lediglich »bei sich unter«. Der Grund ist, dass Sie selber normalerweise nicht über so leistungsstarke und teure Internet-Anbindungen verfügen wie der Provider.

Der Vorteil beim Server-Housing ist gegenüber dem Server-Hosting, dass Sie die Hardware des Rechners völlig nach Ihren Wünschen zusammenstellen können. Ein Nachteil ist, dass der Provider normalerweise keinen Support für Probleme übernimmt, die nichts mit der Internet-Anbindung zu tun haben. Beim Server-Hosting, wo der Provider die Hardware stellt, ist dies noch eher der Fall. Ansonsten gelten für das Server-Housing die gleichen Bemerkungen wie für das Server-Hosting.

### 25.2.8 Publizieren mit FTP

Wenn Sie nicht gerade selbst an der Bedienkonsole eines Server-Rechners sitzen, müssen Sie die Datenverwaltung (Dateien hoch laden, löschen usw.) auf dem Server von einem beliebigen Rechner mit einfachem Internet-Zugang aus erledigen. Das heute übliche Internet-Protokoll dazu ist FTP (File Transfer Protocol). Zur reinen Datenverwaltung (ohne Datei-Upload und -Download) sind die Internet-Dienste Telnet oder SSH noch besser geeignet, da sie praktisch eine »Konsolensimulation« leisten, also das Arbeiten auf dem fernen Server-Rechner erlauben, so als säße man vor dessen Bildschirm.

Von Ihrem Provider erhalten Sie alle erforderlichen Daten, die Sie brauchen, um Ihre Daten auf dem Server verwalten zu können. Was Sie zum Hochladen noch brauchen, ist ein leistungsfähiges FTP-Programm. Im Link-Verzeichnis des Online-Angebots von SELFHTML bei *selfaktuell.teamone.de* aktuell finden Sie Verweise FTP-Programmen.

Die meisten modernen FTP-Programme haben grafische Oberflächen, die bekannten Dateimanager oder Datei-Commandern ähneln. Sie benötigen dabei keine tiefer reichenden Kenntnisse. Zum Zugriff auf den Server-Rechner brauchen Sie die FTP-Adresse Ihres Speicherbereichs, eine User-ID und ein Passwort. Diese Daten teilt Ihnen der Provider mit. Die Daten können Sie bei den meisten modernen FTP-Programmen in einer integrierten FTP-Site-Verwaltung fest speichern.

Einige Tipps noch:

- Wenn Ihr Projekt aus Unterverzeichnissen besteht, benennen Sie die Verzeichnisse auf dem lokalen Rechner, wo Sie das Projekt erstellen, genau so wie auf dem Server-Rechner. So können Sie besser kontrollieren, was wo steht. Außerdem sollten die Verweise und Referenzen innerhalb des Projekts lokal genau so funktionieren wie auf dem Server. Achten Sie auf Groß- und Kleinschreibung bei Datei- und Verzeichnisnamen. Die meisten Server-Rechner laufen unter Betriebssystemen, die Groß-/Kleinschreibung (im Gegensatz zu Windows) unterscheiden. Am einfachsten vermeiden Sie Probleme, indem Sie alle Datei- und Verzeichnisnamen durchweg klein schreiben.
- Wenn Sie einzelne Dateien lokal auf Ihrem Rechner aktualisiert oder hinzugefügt haben, lassen Sie sich die Dateiliste vor dem Hochladen der Dateien auf den Server-Rechner nach Datum sortiert anzeigen. So sehen Sie besser, welche Dateien neu oder erneuert sind, und vergessen nichts beim Hochladen.

- Kontrollieren Sie nach jedem Hochladen von Dateien das Vorgangsprotokoll, das Ihr FTP-Programm aufzeichnet. Wenn dort bei der Übertragung von Dateien irgendwelche Fehler aufgezeichnet sind, laden Sie diese Dateien noch einmal hoch.
- Lassen Sie sich nach jedem Hochladevorgang den aktualisierten Inhalt auf dem Server-Rechner anzeigen und überprüfen Sie, ob die neuen Dateien angezeigt werden. Beachten Sie, dass die Anzeige von Datum und Uhrzeit auf dem Server-Rechner nicht unbedingt Ihrer lokalen Zeit entsprechen muss. Viele Server-Rechner stehen in den USA und benutzen als Systemzeit die dort aktuelle Lokalzeit. Auch bei Dateigrößen können Unterschiede zwischen lokaler Version und Server-Version einer Datei angezeigt werden. Das muss nicht bedeuten, dass die Datei bei der Übertragung beschädigt wurde.

Rufen Sie die neuen Dateien anschließend mit einem Web-Browser auf, um zu kontrollieren, ob alles so funktioniert, wie es soll.

## 25.3 Web-Projekte bekannt machen

### 25.3.1 Allgemeines

Nachdem Sie ein neues Projekt im Web publiziert haben, weiß niemand außer Ihnen und Personen, denen Sie die Adresse persönlich mitteilen, etwas von der Existenz des Projekts. Damit Ihr Projekt tatsächlich auffindbar, und besser noch, bekannt wird, müssen Sie es bekannt machen. Dazu gibt es verschiedene Möglichkeiten, die in diesem Abschnitt beschrieben werden. Bitte beherzigen Sie folgende allgemeine Regeln:

- Seien Sie nicht marktschreierisch. Das mag niemand und erreicht eher das Gegenteil von dem, was Sie sich wünschen.
- Trachten Sie nicht so sehr danach, dass möglichst viele Leute Ihre Seiten besuchen, sondern eher danach, dass die Besucher begeistert sind, sich freiwillig längere Zeit in Ihrem Projekt aufhalten und gerne wiederkommen. Bekanntheit und ein guter Name sind nämlich zweierlei, und es ist besser für das Internet, wenn es Ihnen vor allem auf den guten Namen ankommt.
- Bleiben Sie auf dem Boden, wenn Ihr Projekt tatsächlich bekannter wird. Das Web ist nicht Hollywood. Überheblichkeit und Star-Gehabe werden mit Nichtachtung und Abwendung gestraft.

### 25.3.2 Projekte bei Suchdiensten und Verzeichnissen anmelden

Alle großen Suchmaschinen im Web bieten die Möglichkeit an, neue Seiten anzumelden. Dann bekommen Ihre Seiten nach einer Weile (es kann bei manchen Suchmaschinen wegen notorischer Überlastung leider sehr lange dauern) Besuch von einem Softwareprogramm, einem so genannten Robot. Dieser benimmt sich wie ein eifriger Web-User und folgt allen Links, die er auf der angemeldeten Adresse findet. Alles, was er abgrast, merkt er sich und speichert die Texte der Seiten in der großen Datenbank der Suchmaschine. Von da an sind Ihre Seiten in der Suchmaschine auffindbar. Der Robot kommt dann von Zeit zu Zeit wieder und erfasst neue und geänderte Daten auf Ihren Seiten.



### 25.1: Google ist eine der am häufigsten genutzten Suchmaschinen.

Daneben gibt es auch Web-Verzeichnisse, die redaktionell basiert sind. Melden Sie Ihre Seiten bei einem solchen Verzeichnis an, dann werden die Seiten nach einer Weile von einem Redakteur besucht, der darüber entscheidet, ob die Seiten in das Verzeichnis aufgenommen werden.

Bei folgenden Suchmaschinen und Web-Verzeichnissen sollten Sie deutschsprachige Angebote auf jeden Fall anmelden:

- [www.google.de/](http://www.google.de/)
- [www.fireball.de/](http://www.fireball.de/)
- [www.altavista.de/](http://www.altavista.de/)
- [www.yahoo.de/](http://www.yahoo.de/)
- [web.de/](http://web.de/)
- [www.dino-online.de/](http://www.dino-online.de/)



25.2: Web-Verzeichnisse wie Dino-Online sortieren Seiten nach Kategorien.

Wenn Sie in einer Suchmaschine angemeldet sind, bedeutet das allerdings noch nicht, dass nun sofort die Besuchermassen auf Ihre Seiten strömen. Denn erst einmal müssen die Anwender des Suchdienstes ja nach Begriffen suchen, bei denen Ihre Seiten zu den Suchtreffern gehören, und dann müssen sie unter den angebotenen Suchtreffern Ihre Seiten als besonders aussichtsreich einschätzen. Deshalb sind ja auch ordentliche Meta-Angaben in HTML so wichtig.

Je nach Projekt ist es auch sinnvoll, das Projekt bei kleineren, spezielleren Suchmaschinen und/oder Verzeichnissen anzumelden. So sind beispielsweise fast alle bekannten Branchenverzeichnisse auch im Web vertreten. Wenn Sie einen Firmenauftritt im Web bekannt machen wollen, ist

eine Anmeldung dort durchaus sinnvoll, wenngleich Sie auch dabei nicht zu viel erwarten sollten, zumal die Anmeldung im Gegensatz zu der Anmeldung bei Suchmaschinen Geld kostet. Bei Fach-Projekten lohnt auf jeden Fall die Anmeldung bei Katalogen und fachbezogenen Suchdiensten. Dazu müssen Sie allerdings erst mal wissen, welche Spezialsuchdienste und -Verzeichnisse es überhaupt gibt. Für diesen Zweck gibt es eine Suchmaschine für Suchmaschinen, [www.suchlexikon.de/](http://www.suchlexikon.de/)

Es kann auch sinnvoll sein, Betreiber bekannter anderer Web-Angebote anzuschreiben und höflich anzufragen, ob es dort möglich ist, einen Link auf das eigene Angebot zu setzen. Treten Sie aber nicht so auf, als ob darauf irgendein Anspruch bestünde. Es bleibt dem Ermessen des Anbieters überlassen, ob er einen Link auf Ihre Seiten setzt oder nicht.

### 25.3.3 Strategien bei der Online-Kommunikation

Vermutlich werden Sie das Internet ja auch für E-Mail-Korrespondenz nutzen. Legen Sie sich in Ihrem Mail-Programm eine **Signatur** an, die automatisch am Ende aller Mails eingefügt wird, die Sie schreiben. Die Signatur sollte auch Ihre Web-Adresse enthalten, ansonsten aber unaufdringlich wirken. Ein Beispiel:

```
-----
Rudolf Rabenstein, Hamburg, Tel. 0173-308934023
http://www.rudolf-rabenstein-online.de/
-----
```

Wenn Sie Seiten zu bestimmten Fachgebieten oder Themen haben, empfiehlt es sich auch, an Internet-Diskussionen zu den entsprechenden Themen teilzunehmen. So gibt es für fast alle Gebiete Newsgroups, Mailing-Listen oder webbasierte Foren. Gute Übersichts- und Einstiegsadressen dafür sind:

- [www.mailinglisten.de/](http://www.mailinglisten.de/)
- [www.usenet.de/](http://www.usenet.de/)
- [www.wer-weiss-was.de/](http://www.wer-weiss-was.de/)

Überzeugen Sie in Diskussionen durch fachliche und kommunikative Kompetenz. Benutzen Sie auch in solchen Diskussionsbeiträgen eine Signatur, ähnlich wie in privaten E-Mails. Aber achten Sie dabei erst recht darauf, eine unauffällige Signatur zu verwenden. Alles andere wird als lästige Werbung empfunden. Durch überzeugendes Auftreten in Fachkreisen spricht sich dann auch Ihre Web-Adresse herum.

### 25.3.4 Sonstige Bekanntmachungsstrategien

Bei neuen, professionellen Projekten sind auch **Pressemitteilungen** angebracht. Wenn es sich um computerorientierte Projekte handelt, sind PC-Fachzeitschriften zum Beispiel eine gute Adresse. Wenn es sich um ein Hobby anderer Art handelt (Radfahren, Angeln, Tauchen, Reisen usw.), können Sie eine Pressemitteilung an entsprechende Fachmagazine verfassen. Bei elektronischen Magazinen können Mitteilungen an Life-Style-Magazine erfolgen. Die Pressemitteilung kann per Post, per Fax oder per E-Mail erfolgen. Bei den meisten Medien (Print wie Hörfunk) sind die

Internet-Redaktionen Anhängsel zu den althergebrachten Redaktionen. Es lohnt sich deshalb eher, sich an die traditionellen Redaktionen zu wenden. Bei online-spezifischen Angeboten kann das anders sein. Adressen erfahren Sie aus dem Impressum des jeweiligen Publikationsorgans. Bitte schreiben Sie aber nicht wegen Ihrer neuen Goldhamster-Homepage an die Frankfurter Allgemeine.

Wenn Sie als Unternehmen ohnehin **Anzeigen** in Zeitungen, Zeitschriften, Fachpublikationen oder audiovisuellen Massenmedien schalten, platzieren Sie dort auch die Homepage-Adresse. Oft genügt es, die Web-Adresse ohne jeden Kommentar an einer geeigneten Stelle zu platzieren. Wer weiß, was eine solche Adresse ist, weiß sofort, was gemeint ist, und den anderen können Sie es an der Stelle ohnehin nicht erklären.

Wenn Sie viel Kundenverkehr haben, legen Sie **Handzettel** mit einer Kurzvorstellung Ihrer Homepage aus. Achten Sie auch darauf, die Adresse auf Ihrer **Visitenkarte**, auf Ihren **Briefbögen**, Ihren **Faxvordrucken** usw. mit abzudrucken.

Machen Sie sich im Web schlau, welche weiteren Möglichkeiten es gibt. So gibt es beispielsweise so genannte **Web-Ringe** zu allen nur erdenklichen Themen. Bei einem Web-Ring verlinken sich mehrere Web-Projekte zum gleichen Thema ringartig, sodass ein Besucher über eines der Projekte auch zu den anderen gelangt.

Kaum noch zu empfehlen ist das Schalten eigener **Werbe-Banner** auf fremden Web-Seiten. Die Klickraten bei Werbe-Bannern sind mittlerweile so niedrig, dass aus Anwendersicht die Hoffnung besteht, irgendwann von diesen scheußlichen und jedes Web-Design zerstörenden Dingen befreit zu sein. Die Kosten für das Schalten solcher Banner stehen normalerweise in keinem Verhältnis zu dem tatsächlichen Nutzen.

## 25.4 Web-Projekte kontrollieren

### 25.4.1 Allgemeines

Wenn Sie ein Projekt im Web publiziert haben, werden Sie sicher auch interessiert sein zu erfahren, wie häufig darauf zugegriffen wird. Um richtig einschätzen zu können, wie gut Ihr Projekt ankommt und was Sie daran verbessern sollten, reichen allerdings reine Zugriffszahlen nicht aus. Um aussagekräftige Ergebnisse über die Abrufe Ihrer Web-Seiten zu erhalten, sind Statistiken nötig, die Ihnen die Zugriffe nach bestimmten Kriterien aufschlüsseln. So gibt es Statistiken, die auflisten, welche Dateien am häufigsten abgerufen wurden, zu welchen Tageszeiten die meisten Besucher zu verzeichnen sind oder welche Web-Browser am häufigsten verwendet werden.

Bei besseren Providern gehört es zum Service, dass Sie Zugriff auf Log-Dateien haben, die Ihren Speicherbereich betreffen. Die meisten Provider bieten auch Statistiken an, mit deren Hilfe Sie die Besucherzahlen nach verschiedenen Kriterien aufbereitet bekommen.

Wenn Sie Ihre Homepage bei Discount-Providern wie Compuserve, T-Online usw. oder bei Anbietern von kostenlosem Webspace haben, stehen Ihnen solche Dienste meist nicht zur Verfügung. Um in solchen Fällen an genaue Zugriffsstatistiken zu kommen, können Sie einen öffentlichen CGI-Service in Anspruch nehmen. Dort bekommen Sie für eine geringe Jahresgebühr einen Zugriffszähler sowie ordentlich aufbereitete Statistiken für Ihre Homepage, egal wie

deren Adresse lautet. Ein Nachteil bei dieser Lösung ist allerdings, dass bei jedem Zugriff auf Ihre Homepage zusätzlich auf den fremden Server des Zähler-Anbieters zugegriffen werden muss.

## 25.4.2 Log-Dateien des Web-Servers

Grundlage aller Zugriffsstatistiken sind Log-Dateien des Web-Servers. Jeder Web-Server führt solche Log-Dateien, in denen er alle Zugriffe protokolliert. Dazu gibt es ein allgemeines Modell, wie Einträge in einer Log-Datei auszusehen haben. Ein Eintrag nach diesem Modell hat folgendes Aussehen:

```
ad25-175.compuserve.com - - [02/Oct/2001:23:03:49 -0500] "GET /selfhtml/index.htm
HTTP/1.1" 200 1788
```

Jede Zeile in der Log-Datei dokumentiert einen Zugriff auf eine Datei. Das Beispiel zeigt eine solche Zeile. Log-Dateien viel besuchter Projekte enthalten Zigtausende solcher Zeilen und können sehr schnell sehr groß werden. Meistens sorgen regelmäßig automatisch startende Server-Prozesse, so genannte Cron-Jobs, dafür, dass die Log-Dateien regelmäßig archiviert und wieder geleert werden. Die Log-Dateien haben meistens Namen wie *access.log*. Wo diese Dateien abgelegt sind, hängt von der Server-Konfiguration ab. Fragen Sie Ihren Provider danach, sofern er Ihnen von sich aus keine Angaben dazu macht!

Am Beginn jeder Zeile steht, wer zugegriffen hat. Bei den meisten Zugangs-Providern erhalten die Anwender beim Einwählen ins Internet eine dynamische IP- und Namensadresse, unter der sie dann in den Log-Dateien der Web-Server auftauchen, die sie besucht haben. Im obigen Beispielintrag ist erkennbar, dass sich der Anwender über CompuServe ins Internet eingewählt hat. Die Angabe *ad25-175* ist eine dynamisch vergebene Adresse.

Anstelle der beiden Bindestriche im obigen Beispiel können von Fall zu Fall noch Daten stehen. Anstelle des ersten Bindestriches kann der Anwender-Logname stehen (entsprechend der Spezifikation RFC931). Anstelle des zweiten Bindestriches kann ein Anwendername stehen, den der Anwender sich selbst gegeben hat.

Als Nächstes zeigt der Eintrag, wann genau der Zugriff erfolgte. Im Beispiel ist das der 2. Oktober 2001 um 23.03 Uhr (und 49 Sekunden), und zwar minus 5 Stunden gerechnet gegenüber GMT (Greenwich Meridian Time) aus Sicht der Systemzeit des Server-Rechners.

Dahinter folgt ein Kurzprotokoll des HTTP-Headers, mit dem der WWW-Browser des Anwenders die Datei angefordert hat. *GET* ist dabei die übliche Methode. Dahinter wird die Datei angegeben, auf die der Zugriff erfolgte. Im Beispiel ist das die Datei */selfhtml/index.htm*. Es wird der Pfadname relativ zur Domain-Adresse angezeigt. Schließlich zeigt die HTTP-Anforderung noch die verwendete HTTP-Protokollversion an. Das war zur Auswertungszeit die Version 1.1.

Dahinter folgt eine Codenummer, die darüber Aufschluss gibt, ob die Anforderung erfolgreich war, und wenn nicht, welches der Fehler war. Dabei sind folgende Nummern wichtig:

- 200 = Die Anforderung war erfolgreich.
- 4xx, 5xx = Es traten Fehler auf. Zum Beispiel war die angeforderte Datei nicht vorhanden. Lesen Sie dazu auch die Übersicht der HTTP-Statusmeldungen in Anhang A.1.

Die letzte Zahl in der Zeile schließlich gibt an, wie viele Bytes von der Datei an den aufrufenden WWW-Browser übertragen wurden. Wenn Sie die Bytezahl mit der tatsächlichen Dateigröße

vergleichen, können Sie feststellen, ob die Datei vollständig übertragen, oder ob die Übertragung vorzeitig abgebrochen wurde. So kommt es zum Beispiel oft vor, dass Grafiken nicht vollständig übertragen werden, weil der Anwender bereits vorher auf einen Verweis zu einer anderen Seite klickt.

Neben den reinen Zugriffs-Log-Dateien führen Web-Server auch Log-Dateien über aufgetretene Fehler. Solche Log-Dateien haben Namen wie *error.log*. Die Dateien sind ähnlich aufgebaut wie Zugriffs-Log-Dateien. Das folgende Beispiel zeigt eine typische Zeile aus der Error-Log-Datei des Web-Servers Apache:

```
[Tue Oct 02 23:04:34 2001] [error] [client ad25-175.compuserve.com] File does not exist: /selfhtml/tc.htm
```

Zu Beginn steht ein Zeitstempel im UTC-Format. Im Beispiel ist Dienstag (*Tue* = *Tuesday* = *Dienstag*), der 2. Oktober 2001, 23.04 Uhr und 34 Sekunden gemeint. Dahinter wird das Vorzeichen genannt (*error*) und dahinter der verursachende Client (z.B. der Browser eines Seitenbesuchers). Die Fehlermeldung wird in englischem Klartext genannt. Im Beispiel hat der Anwender versucht, auf eine Datei zuzugreifen, die nicht existiert. Schuld daran kann beispielsweise ein Link von einer fremden Seite sein, der zu dieser Adresse führt, die aber nicht oder nicht mehr existiert.

### 25.4.3 Statistiken

Log-Dateien sind normalerweise sehr groß und etwa so spannend wie ein Telefonbuch. Um eine aussagekräftige Übersicht über die Zugriffe auf Ihre Seiten zu erhalten, sollte Ihr Provider Statistiken anbieten. Dazu bietet Ihnen der Provider bei Virtual Hosting in der Regel einen über Web-Browser aufrufbaren Service-Bereich an, zu dem Sie nur mittels eines Benutzernamens und Passworts Zugang haben. Bei Server Hosting oder Server Housing kann es auch sein, dass Sie sich selber um die Installation geeigneter Statistik-Software kümmern müssen.

Statistiken bestehen einerseits aus Software, die aus den Log-Dateien regelmäßig ihre eigene Datenbank pflegt, und einem oder mehreren CGI-Scripts, die Sie aufrufen können, um die Statistik im Browser in HTML-Form aufbereitet ansehen zu können. Es gibt rein textorientierte Statistiken, die Daten in Listen oder Tabellen präsentieren, aber auch grafische Statistiken, bei denen die Daten in Form von Balken- und Tortendiagrammen präsentiert werden.

Statistiken stellen Ihnen die Zugriffe nach einem bestimmten Kriterium sortiert dar. Folgende Kriterien sind typisch:

- Auflistung nach Dateien (auf welche Dateien wurde am häufigsten zugegriffen?)
- Auflistung nach Zeit (zu welcher Tageszeit, an welchen Wochentagen wurden die meisten Zugriffe verzeichnet?)
- Auflistung nach Herkunft (über welche Domain-Adressen haben sich die meisten Anwender ins Internet eingewählt?)
- Auflistung nach Erfolg (welche Dateien wurden vollständig übertragen, bei welchen wurde bei der Übertragung vorzeitig abgebrochen?)
- Auflistung nach Anwendersoftware (welche WWW-Browser in welchen Versionen wurden am häufigsten verwendet?)



Statistiken bleiben allerdings stets nur relativ genau. Absolute Genauigkeit ist deshalb nicht möglich, weil Statistiken letztlich nur auf den Log-Dateien des Web-Servers basieren, es aber viele andere Faktoren gibt, die das Abrufen von Web-Seiten mit beeinflussen. So zum Beispiel der Cache-Speicher eines jeden Web-Browsers. Je nachdem, was der Anwender dort eingestellt hat, wird die Web-Seite, obwohl vom Anwender aufgerufen, gar nicht vom Web-Server geholt, sondern aus dem Browser-Cache, ohne dass der Browser wenigstens noch mal beim Server nachfragt, ob es von der Web-Seite mittlerweile eine neuere Version gibt als die in seinem Cache. Der Web-Server bekommt folglich gar nichts davon mit. Ebenso ist es mit den zahlreichen Proxy-Servern im Netz. Viele Anwender haben im Browser freiwillig oder auf Anforderung ihres Providers hin Proxy-Server eingestellt. Proxy-Server sind für den Internet-Zugang des Anwenders besonders leicht und ohne viele Umwege erreichbar, und das Übertragen der Daten von dort aus zum Browser geht merklich schneller, als wenn die Daten vom Original-Server übertragen werden. Wenn Ihr Web-Projekt auf einem solchen Proxy-Server verfügbar ist (jawohl, ihr Web-Projekt wird von solchen Proxy-Servern automatisch komplett »geklaut« und zur Verfügung gestellt – das ist ganz normal im Internet), ist der Effekt der gleiche wie beim Browser-Cache: Der Web-Server Ihres Projekts bekommt gar nichts davon mit. Andererseits werden in der Log-Datei des Web-Servers aber auch viele Einträge erzeugt, die gar nicht von menschlichen Besuchern stammen. Such-Robots von Suchmaschinen, die Ihre Seiten indizieren, oder Software-Agenten von Proxy-Servern, die Ihre Seiten an den Proxy-Server übertragen – sie alle erzeugen »Traffic«, also Verkehr auf Ihren Seiten, ohne dass ein Mensch daran beteiligt ist.

#### 25.4.4 Statistiken auswerten

Die folgende Checkliste soll Ihnen helfen, Statistiken zu interpretieren und Schlüsse daraus zu ziehen:

- **Haben Sie so gut wie keine Zugriffe?** Wenn Ihr Projekt erst seit wenigen Tagen im Netz steht, ist das normal. Wenn es allerdings schon seit Monaten online ist, haben Sie zu wenig unternommen, um das Projekt bekannt zu machen. Die Zugriffszahlen hängen natürlich auch von den angebotenen Themen ab. Ein Projekt über die neuesten Computerspiele hat nun einmal mehr Zugriffe als ein Projekt über seltene Orchideensorten.
- **Haben Sie viele Zugriffe auf die Einstiegsdatei, jedoch so gut wie keine auf die Unterseiten?** Es ist ganz normal, dass die Einstiegsseite mit Abstand die meisten Zugriffe zu verzeichnen hat. Wenn aber die Unterseiten so gut wie gar nicht besucht werden, deutet das darauf hin, dass vielen Besuchern Ihre Einstiegsseite nicht gefällt, dass bei der Einstiegsseite zu viele Daten (z.B. aufwendige Grafiken, Flash-Intros usw.) übertragen werden, oder dass die Einstiegsseite so unübersichtlich ist, dass die Besucher die Verweise zu den Unterseiten nicht finden. In diesem Fall sollten Sie Ihre Einstiegsseite neu, einfacher und übersichtlicher gestalten und überprüfen, ob das Verhältnis anschließend günstiger wird.
- **Werden große Grafiken häufig genug vollständig übertragen?** Das gilt besonders für Einstiegsseiten. Wenn eine solche Seite ihren Blickfang beispielsweise aus einer 50 Kilobyte großen Grafik bezieht und von dieser Grafik im Schnitt nur die Hälfte übertragen wird, bedeutet das, dass kaum ein Besucher darauf wartet, bis die Seite ihre ganze Pracht entfaltet hat. In diesem Fall sollten Sie die Seite dringend neu gestalten.
- **Wie lange halten sich Besucher auf Ihren Seiten auf?** Das können Sie am besten direkt an den Logdateien erkennen. Wenn Sie 50 Seiten haben und die Besucher im Schnitt unter einer

Minute verweilen, ist das Projekt einfach nicht attraktiv genug. Ausgenommen davon sind Projekte mit dem Charakter von Nachschlagewerken, in denen niemand stöbert, sondern nur eine gezielte Auskunft sucht. Sie können nicht erwarten, dass die Besucher stundenlang bleiben und alles von vorne bis hinten studieren. Aber mehrere Minuten im Schnitt deuten darauf hin, dass die Besucher sich auch mal »verführen« lassen.

- **Wo kommen Ihre Besucher her?** Achten Sie dabei darauf, von welchen fremden Adressen aus auf Ihr Projekt zugegriffen wird. So können Sie beispielsweise erkennen, ob Sie häufig über Themensuchen in Suchdiensten gefunden werden oder ob eine anklickbare Anzeige, die Sie bei einem Kataloganbieter gegen Gebühr geschaltet haben, sich rentiert. Auch auf die Länderherkunft sollten Sie achten. Wenn Sie viele Besucher aus dem fremdsprachigen Ausland haben, aber kein mehrsprachiges Angebot, dann ist zu überlegen, ob ein mehrsprachiges Angebot die Attraktivität des Projekts nicht vervielfachen könnte.

### 25.4.5 Begriffe wie Hits, Pageviews und Visits

Statistiken sind bekanntlich dehnbar. Es kommt immer darauf an, was eigentlich gezählt wird. Deshalb ist es wichtig, dass Konventionen darüber bestehen, welche typischen Begriffe welche Bedeutung haben. Denn wann ist ein Zugriff ein Zugriff? Sobald der Web-Server die GET-Anfrage des Clients erhalten hat? Oder nachdem die angeforderte Web-Seite vollständig an den aufrufenden Client, also beispielsweise den Browser des Anwenders, übertragen hat? Mit oder ohne referenzierte Dateien wie Grafiken, Sound usw.?

Jede Zeile im Access-Log des Web-Servers erzeugt einen so genannten **Hit**. Wenn eine Web-Seite also beispielsweise aus einer HTML-Datei und 10 darin referenzierten Grafiken besteht, dann erzeugt diese Web-Seite, wenn ein Anwender sie anfordert, 11 Hits. Dabei ist es egal, ob die Dateien vollständig übertragen wurden oder nicht. Sogar nicht gefundene Dateien (404-Errors) werden als Hits gezählt. Hits liefern also hohe, beeindruckende Zahlen, die sich aber schnell relativieren, wenn man sich klar macht, dass viele typische Web-Seiten etliche kleine, referenzierte Grafiken enthalten. Außerdem geht aus den Hits nicht hervor, ob der Besucher die Seite je zu Gesicht bekommen hat. Lassen Sie sich also nicht beeindrucken von Leuten, die mit ihren Hits prahlen. Für die Einschätzung des tatsächlichen Erfolgs einer Web-Seite sind Hits nicht geeignet. Interessant sind die Hits eigentlich nur im relativen Vergleich, z.B. um herauszufinden, wie sich der Gesamtverkehr auf den Web-Seiten über die Monate hinweg entwickelt.

**Pageviews** – neuerdings auch **Page-Impressions**, bei eingedeutschten Statistiken auch **HTML-Seiten** oder **Seiten-Anfragen** genannt – sind dagegen »Sichtkontakte eines Anwenders mit inhaltstragenden Dateien«. Zugriffe von Anwendern, die in ihrem Browser die Anzeige von Grafiken abgeschaltet haben, sowie Zugriffe von Such-Robots und anderen Software-Agenten werden dabei, falls erkennbar, nicht mitgezählt. Bei Verwendung von Frames wird beim Erstaufruf nur die Datei als Pageview gezählt, in der die Frameset-Definitionen stehen. Danach wird davon ausgegangen, dass nur eines der Frame-Fenster den Inhalt beherbergt, während die übrigen zur Navigation oder für Werbung und Ähnliches dienen. Die so definierte statistische Messeinheit ist die Kenngröße der Informationsgemeinschaft zur Festlegung der Verbreitung von Werbeträgern e.V. (IVW – siehe [www.ivw.de/](http://www.ivw.de/)). Pageviews oder Page-Impressions sind daher die Basis, wenn Sie z.B. Bannerwerbung auf Ihren Seiten anbieten und Geld dafür bekommen. Völlig unproblematisch ist diese Maßeinheit jedoch auch nicht. Denn zunächst mal ist variabel, welche Dateitypen als inhaltstragend deklariert werden. Das können je nach Einstellungen statische HTML-Dateien,

PHP- oder ASP-Dateien, Textdateien, XML-Dateien oder auch CGI-Scripts sein. Eine HTML-Datei kann inhaltstragend sein, sie kann aber auch nicht inhaltstragend sein (z.B. weil sie nur Links zu Unterseiten enthält, nur eine Weiterleitung darstellt oder nur ein eingebettetes Flash-Intro enthält). HTML-Dateien mit Includes (SSI, PHP, ASP) können Inhalte aus mehreren Inhaltsquellen in einer Seite zusammenfassen. Andere Lösungen benötigen zur Anzeige des gleichen Inhalts mehrere HTML-Dateien oder Scripts.

**Visits** – auch **Sessions** oder in deutschsprachigen Statistiken **Besuche** oder **Sitzungen** genannt – zählen nicht aufgerufene Seiten, sondern die IP-Adressen aufrufender Clients. Ein Anwender, der sich eine Stunde lang munter durch das Angebot Ihrer Web-Seiten klickt und dabei ununterbrochen mit der gleichen Internet-Verbindung unterwegs ist, erzeugt demnach genau einen Visit. Ebenso aber auch ein Anwender, der nur mal die Startseite aufruft und dann gleich wieder verschwindet. Per Konvention der deutschen Werbeindustrie gilt ein Visit als beendet, wenn 30 Minuten lang kein neuer Zugriff erfolgt. Wenn der Anwender also während Ihre Seiten auf seinem Browser angezeigt werden eine längere Mittagspause einlegt oder einschläft und erst nach zwei Stunden auf den nächsten Verweis in Ihrem Angebot klickt, dann gilt das als neuer Visit.

### 25.4.6 Anwender-Feedback auswerten

Wie Ihre Web-Seiten bei den Besuchern ankommen, erfahren Sie nicht nur aus Statistiken, sondern beispielsweise auch aus E-Mails, die Sie von Anwendern erhalten, oder aus den Einträgen eines Gästebuchs, das Sie auf Ihren Web-Seiten anbieten. Es ist ratsam, alles erhaltene Feedback in irgendeiner Form zu sammeln und sinnvolle Verbesserungsvorschläge, Korrekturen usw. ins Angebot zu integrieren.

Wenig Anwender-Feedback trotz guter Zugriffszahlen muss nichts Negatives bedeuten. Ein ordentlich gestaltetes, den Anwendererwartungen entsprechendes Angebot wird dankbar genutzt, ohne dass die Anwender deswegen auf die Idee kommen, Ihnen begeisterte Mails zu schreiben. Explizites Feedback erhalten Sie eher, wenn Ihr Angebot in welcher Form auch immer aus dem Rahmen fällt: Der Grund kann sein, dass Anwender enttäuscht sind, weil das Angebot nicht ihren Erwartungen entspricht, oder dass Anwender hin und weg sind, weil das Angebot ihre Erwartungen bei weitem übertrifft und ihnen ganz neue Perspektiven eröffnet.

Ein möglicher Grund für erhöhtes Anwender-Feedback sind technische Besonderheiten Ihrer Seiten. Wenn Ihre Navigation beispielsweise nur mit eingeschaltetem JavaScript und nur mit dem neuesten Internet Explorer funktioniert, dann werden Sie zu Recht einige erboste Reaktionen von Anwendern erhalten, bei denen diese Voraussetzungen nicht gegeben sind. Ebenso ist es mit Seiten, die bestimmte Bildschirmauflösungen erzwingen, den Anwender durch »Features« wie Sperren der rechten Maustaste nerven und dergleichen. Umgekehrt kann es aber auch passieren, dass Anwender sich melden, weil Ihr Seiten-Layout ungewöhnlich gut ist oder weil bestimmte technische Features, die sie sonst noch nirgends gesehen haben, ihr Interesse geweckt hat. Negatives Feedback zur Technik Ihrer Seiten sollten Sie auf jeden Fall ernst nehmen, zumindest dann, wenn es von mehreren Seiten und immer wieder geäußert wird.

Wenn Sie echte Informationen vermitteln, werden Sie auch Fachfragen von Anwendern erhalten. Wie Sie damit umgehen, ist Ihre Sache. Jedenfalls können Sie solches Feedback als Indiz dafür betrachten, dass Ihr Angebot genutzt wird, dass die Informationen gelesen werden und dass die Anwender Sie für fachkompetent halten.

Spätestens wenn Sie regelmäßiges Feedback von den gleichen Anwendern erhalten, haben Sie »Fans«. Wenn Sie bis dahin nur ein reines Informationsangebot auf Ihren Seiten haben, sollten Sie darüber nachdenken, ob das Angebot durch Anbieten von Kommunikationsmöglichkeiten wie Web-Forum oder Chat-Raum nicht noch viel attraktiver werden könnte. Mit etwas persönlichem Engagement können Sie es dann schaffen, dass über Ihr Web-Projekt eine richtige Online-Community entsteht – was wohl der Traum vieler Web-Anbieter ist.

## 25.5 Provider wechseln mit Web-Projekten

### 25.5.1 Provider wechseln bei vorhandener Domain-Adresse

Ein mittlerweile häufiger Fall ist, dass Sie bereits eine eigene Domain-Adresse haben, also z.B. eine Adresse vom Typ *IhrName.com* oder *IhrName.de*, und nun den Provider wechseln möchten. Die Adresse möchten Sie aber behalten. Das ist normalerweise kein Problem, und eine vorhandene Domain-Adresse können Sie bei einem Wechsel zu einem anderen Provider einfach »mitbringen«. Die numerische Rechneradresse, auf die Ihre Seitenbesucher fortan geroutet werden, muss dann geändert werden, aber die Namensadresse, unter der Ihr Projekt bekannt ist, bleibt die gleiche.

Für den Wechsel müssen Sie – zumindest bei *de*-Domains – einen so genannten **KK-Antrag** ausfüllen, in dem Sie schriftlich und mit Unterschrift als verantwortlicher Domain-Inhaber (»admin-c«) den Änderungswunsch bekunden (KK steht für »Konnektivitäts-Koordination«). Bei anderen Top-Level-Domains (*.com*, *.net* usw.) gelten andere Regeln. Fragen Sie am besten den neuen Provider danach! Dieser startet die **Umkonnektierung**. Dabei werden die Routing-Tabellen auf allen Gateway-Rechnern im Internet geändert, die dafür sorgen, dass weltweit alle Aufrufe mit Ihrem Domain-Namen auf den Rechner geroutet werden, auf dem Ihre Daten nun abgelegt sind. In einem gewissen Übergangszeitraum nach der Umkonnektierung kann es zu technischen Problemen kommen, an denen weder der alte noch der neue Provider Schuld sind. Denn ein Teil der Seitenbesucher wird dann noch zu dem alten Server-Rechner geroutet, ein anderer Teil bereits zu dem neuen. Diese Probleme müssen Sie einfach in Kauf nehmen – sie erledigen sich nach einer Weile von alleine.

### 25.5.2 Provider wechseln und neue Domain-Adresse

Dieser Fall tritt ein, wenn Sie bereits ein Web-Projekt haben, aber bislang keine eigene Domain-Adresse, z.B. weil Sie das Projekt bei einem Online-Dienst oder einem kostenlosen Anbieter gehostet haben und nun aber »Nägel mit Köpfen« machen wollen. In diesem Fall müssen Sie erst mal eine eigene Domain-Adresse reservieren. Ihr neuer Provider hilft Ihnen dabei und kann für Sie herausfinden, ob der gewünschte Name noch frei ist.

Nach der **Konnektierung** ist die neue Adresse dann weltweit erreichbar, und Sie können Ihre Projektdateien auf den neuen Rechner hoch laden. Damit nun aber die Besucher auch zu der neuen Adresse finden, müssen Sie jedoch noch ein paar Dinge regeln:

- Fragen Sie Ihren alten Provider, ob es möglich ist, Ihre Daten wenigstens noch ein paar Wochen lang unter der alten Adresse zu stehen zu lassen. Dort können Sie dann anstelle der ehemaligen Einstiegsseite eine einfache Seite platzieren, die einen Verweis auf die neue

Adresse enthält. Dazu brauchen Sie keine opulenten Grafiken. Es genügt ein einfacher Hinweis auf den Umzug und ein gut sichtbarer, anklickbarer Verweis oder eine automatische Weiterleitung auf die neue Adresse.

- Wiederholen Sie alle Aktivitäten zum Bekanntmachen Ihres Projekts. Melden Sie Ihre neue Adresse wieder bei allen wichtigen Suchdiensten und Web-Verzeichnissen an. Falls die Suchdienste eine Option zum Löschen von Einträgen anbieten, machen Sie davon Gebrauch und löschen Sie den Eintrag, der zu der alten Adresse führt. Ändern Sie Visitenkarten, Brief- und Faxvordrucke ab, damit nur noch Ihre neue Adresse in Umlauf gerät.
- Informieren Sie andere Seitenanbieter, von denen Sie wissen, dass deren Projekte Verweise auf Ihr Projekt enthalten. Bitten Sie die Anbieter, die Verweise bei Gelegenheit zu aktualisieren. Drängeln Sie aber nicht und schimpfen Sie nicht gleich auf den fremden Anbieter, wenn der Verweis nicht nach zwei Tagen aktualisiert ist.

Diese Maßnahmen gelten auch, wenn Sie nicht den Provider wechseln, aber ein separates Projekt in einem Unterverzeichnis Ihrer eigenen Domain-Adresse haben und das Verzeichnis umbenennen oder das Projekt an einer ganz anderen Stelle publizieren.

## 25.6 Rechtliche Aspekte von Web-Projekten

### 25.6.1 Allgemeines

Zuerst kamen die Idealisten, dann die Prokuristen, und zuletzt die Juristen ins Web. Was zuerst ein Paradies für Freaks war, wurde dann ein Paradies für Investoren, und schließlich ein Paradies für freiherrliche Serienbriefe. »Abgesahnt durch Abgemahnt« war das Motto, und der deutsche Staat und sein Rechtssystem sahen dem munteren Gelderpressen tatenlos zu oder fanden es sogar prima. Zugegeben – das ist natürlich nur ein sehr spezieller Blick auf das Thema »Recht im Internet«, ausgelöst durch jene SELFHTML-eigene Geschichte, die Sie bei [selfaktuell.teamone.de](http://selfaktuell.teamone.de) im Online-Angebot von SELFHTML aktuell unter dem Titel *Recht+Links* nachlesen können.

Je enger es jedenfalls im Netz wird, desto häufiger streiten sich zwei. Und je mehr Anwender das Web für sich entdecken, desto mehr Anstößliches wird darin gefunden. Für Web-Anbieter ist es mittlerweile leider ratsam, vor dem Publizieren eines Web-Projekts eine Rechtsschutz-Versicherung abzuschließen und sich schon mal nach guten, auf Internet-Recht spezialisierten Anwälten umzusehen, die im Ernstfall schnell eingeschaltet werden können. Allzumal in Deutschland, wo die so genannte »Erkundungspflicht« des Rechtssubjekts als sehr hoch eingeschätzt wird. Je mehr Anwender andererseits in der Lage sind, eigene Web-Seiten zu erstellen, desto mehr extreme Inhalte gelangen ins Netz. Nazischund und Rufmord, Kinderpornos und die sprichwörtlichen Anleitungen zum Bombenbau – auch Menschen mit solchen Absichten lernen HTML, und meistens noch einiges mehr, denn sie sind kommunikativ bestens organisiert und verstehen sich auf blitzschnelle Server-Wechsel und andere Dinge, die Fahndern das Leben schwer machen.

Juristische Aspekte gehören also mittlerweile zum Web Publishing genauso dazu wie Aspekte des Web-Designs oder die Wahl des richtigen Providers. Derzeit ist noch Manches unklar, umstritten und gesetzlich ungeregelt. Noch urteilen Richter über Netzstreitigkeiten, die selber nie eine Computermaus in der Hand hatten, und Politiker, denen es genauso geht, nehmen keinen

Anstoß daran. Aber neue Generationen rücken nach, zum Teil schon geschaffene oder noch zu schaffende Reglements werden sich durchsetzen, und die Rechtssicherheit wird wachsen. Das wird aber zur Folge haben, dass man zum Publizieren im Web ein gewisses juristisches Grundwissen benötigen wird, genauso wie man zum Autofahren die Verkehrsregeln lernen muss. Dieses Kapitel kann nur einen sehr groben Überblick über das Thema aus Sicht eines Nicht-juristen bieten. Der Blickwinkel ist dabei der des Web-Autors.

### 25.6.2 Hoheit und Verantwortlichkeit

Das Web ist ein weltweites Medium, aber es gibt keine weltweiten Gesetze dafür. Eine Web-Seite ist unter ein und demselben URI in Deutschland, in den USA und in China erreichbar. Was im einen Land als Kapitalverbrechen gilt, ist im anderen Land ein Kavaliersdelikt. Worauf im einen Land die Todesstrafe steht, das wird im anderen Land mit Bewährungsstrafen geahndet. Ganz unterschiedliche Moralvorstellungen und religiöse Gebräuche haben ganz unterschiedliche juristische Auswirkungen. Was im einen Land als lahmer Stammtischwitz empfunden wird, kostet im anderen Land ein paar Jahre Freiheitsentzug wegen Volksverhetzung. Da erhebt sich zwangsläufig die Frage: darf jeder alles ahnden, was nach seinem Recht nicht rechtens ist? Wenn es so wäre, dann wäre praktisch jedes Web-Angebot aus Sicht irgendeines Blickwinkels der Welt strafrechtlich oder sittenrechtlich belangbar. Das Web wäre sofort tot. Aber wenn das nicht akzeptabel ist, ist es denn umgekehrt? Kann ein Web-Angebot juristisch nicht belangbar sein, nur weil das, was ihm zum Vorwurf gemacht wird, an irgendeinem anderen Ort der Welt, wo das Angebot aufrufbar ist, juristisch nicht geahndet wird? Wohl auch nicht. Also wird ein Web-Angebot nach dem Recht eines bestimmten Landes beurteilt, nämlich nach dem Recht des Landes, dessen Staatsbürger der Verantwortliche des Web-Angebots ist, und bei nicht natürlichen juristischen Personen nach dem Recht des Landes, in dem diese juristische Person ihren Erfüllungsort hat. Das klingt vernünftig, löst aber nicht unbedingt alle Probleme. Ein Amerikaner, der in deutscher Sprache Nazi-Schund ins Web bringt, erreicht vor allem ein deutsches Publikum, ist aber von Deutschland aus nicht belangbar. Also genügt es für die deutsche Nazi-Szene, ein paar willige Amerikaner zu finden, die ihre Inhalte ins Web bringen, und schon steht jeder in Deutschland als noch so sittenwidrig empfundene Inhalt ganz legal im Web. Nun kann man für solche, eine ganze Volksseele zum Kochen bringen Fälle noch zwischenstaatliche Abkommen schaffen. Aber bei weniger spektakulären, zum Beispiel zivilrechtlichen oder wirtschaftsrechtlichen Angelegenheiten, bestehen ja die gleichen Probleme, und so viele zwischenstaatliche Abkommen, wie dazu nötig wären, kann es niemals geben. Kurzum: Es bleiben Grauzonen, ähnlich wie bei der Hochseeschifffahrt, und im Vorteil ist, wer diese Grauzonen zu nutzen weiß.

Ebenso wie die Frage der Hoheit stellt sich die Frage der Verantwortlichkeit. Wenn eine Web-Seite von einem Server-Rechner im Internet zum Browser eines Anwenders übertragen wird, sind daran letztendlich mehrere juristische Personen beteiligt: einmal der Autor der Web-Seite, dann der verantwortlich Publizierende (bei größeren Web-Projekten können das durchaus verschiedene Personen sein), dann der Provider, der den Server zur Verfügung stellt, dann die Backbone-Provider und Netzbetreiber, über deren Leitungen die Daten transportiert werden, und schließlich der Anwender, der die Seite willentlich aufgerufen hat.

In den Anfangszeiten des Web war die Verteilung der Verantwortlichkeit auf diese verschiedenen juristischen Personen noch nicht klar geregelt. So gab es Fälle, in denen Provider für rechtswidrige Inhalte verklagt wurden, von denen sie gar nichts wussten. Mittlerweile gibt es in Deutschland Gesetze, in denen die Rechte und Pflichten der beteiligten juristischen Personen geregelt

werden. Doch diese Gesetze sind nach Ansicht mancher Fachleute noch nicht optimal aufeinander abgestimmt und lassen daher noch viele Fragen offen.

### 25.6.3 Relevante Rechtsbereiche (Deutschland)

Folgende Rechtsbereiche können für Publikationen im Web relevant sein. Die Angaben beziehen sich auf deutsches Recht.

#### ***Strafrecht***

Inhaltsverantwortliche Anbieter von Web-Seiten sind in Deutschland »Verbreiter von Schriften«. Daher können Web-Autoren und Redaktionsverantwortliche strafrechtlich belangt werden

- durch Verletzung der öffentlichen Ordnung, insbesondere Volksverhetzung (§130 StGB) und Anleitung zu Straftaten (§ 130a StGB),
- durch Verstoß gegen Pornographiegesetze (§ 184 StGB),
- durch Beleidigung, üble Nachrede und Verleumdung (§ 185-188 StGB),
- durch Anwendung von Nebenstrafrecht, das sich aus dem Urhebergesetz, dem Markengesetz, dem Patentgesetz, dem Gesetz des unlauteren Wettbewerbs und dem Datenschutzgesetz ergeben kann.

Abstufungen wie »vorsätzlich«, »fahrlässig« usw. gelten dabei wie üblich und werden von Fall zu Fall bewertet. Es sind zwar im Grunde nur wenige typische strafrechtliche Verstöße, die relevant sind, doch die entsprechenden Gesetze haben ein relativ weites Anwendungsfeld. Hinter der »Anleitung zu Straftaten« verbirgt sich letztlich das halbe Strafgesetzbuch – unvorsichtige Bemerkungen aus Zorn und Empörung, in Dateiform gespeichert und im Web veröffentlicht, können also schnell gefährlich werden. Auch der in § 184 definierte Jugendschutz kann schnell zum Problem werden. Solange es dazu keine anderen oder einschränkenden Bestimmungen für Web-Anbieter gibt, ist er anwendbar, und jedes Web-Angebot, das nicht nachweisen kann, effiziente Abschirmungsmittel einzusetzen, ist bei Nacktdarstellungen oder obszönen Texten – selbst wenn sie künstlerischer Natur sind – potenziell angreifbar.

#### ***Zivilrecht***

Das Zivilrecht umfasst Bürgerliches Recht, Handels-, Wettbewerbs- und Urheberrecht. Wer im Web publiziert, verteilt im Sinne des Zivilrechts Information. Damit werden vor allem folgende typische Unterbereiche des Zivilrechts auf Inhalte von Web-Seiten anwendbar:

- das Urheberrecht (Verstöße gegen fremdes geistiges Eigentum),
- das Marken- und Patentrecht (Verletzungshandlungen gegen registrierte Marken und Patente),
- das Wettbewerbsrecht (Verstöße gegen Regelungen zur vergleichenden Werbung, gegen berufsspezifische Werbeverbote, gegen Preisbindungsregelungen usw.),
- das Vertragsrecht (relevant bei Verkaufsabschlüssen, Mietabschlüssen usw. via Internet),

- das Persönlichkeitsrecht (Verstöße wie Rufschädigung, Veröffentlichung privater Schriften, darunter auch E-Mails),
- das Produkthaftungsrecht (Haftung bei Falschinformationen, die nachweislich Schäden verursachen).

Gefährlich für Web-Anbieter sind im Bereich des Zivilrechts vor allem die Rechte kommerzieller Unternehmen. Größere Firmen unterhalten Rechtsabteilungen, die immer häufiger auch im Internet nach Verletzungshandlungen gegen Schutzansprüche dieses Unternehmens schnüffeln, zumal in unserem Medienzeitalter der wirtschaftliche Erfolg vieler Unternehmen auch immer stärker an deren Image und Marken hängt. Es gibt jedoch große Unterschiede. Während einige Unternehmen viel erdulden nach dem Motto »Hauptsache man redet über uns«, gehen andere vehement gegen Verletzungshandlungen vor, die man ebenso gut als kostenlose Werbung für Produkte des Unternehmens interpretieren könnte. Die Spielräume sind da leider sehr groß, und deshalb ist der Web-Anbieter gerade in diesem Rechtsbereich einer Art Willkür ausgesetzt, der nur mit Rechtsschutzversicherung und guten Kontakten zu kompetenten Anwälten zu begegnen ist.

### Öffentliches Recht

Das öffentliche Recht umfasst in Deutschland das Verfassungsrecht und das Verwaltungsrecht. Für Web-Anbieter gibt es in diesem Bereich weniger typische Anwendungsfälle. Zwar enthält das Grundgesetz Passagen, die sich in Rechtsstreitigkeiten über Web-Inhalte auswirken können. Doch in der Regel sind die Artikel des Grundgesetzes zu allgemein, um in einem konkreten Rechtsstreit entscheidende Anwendungsschlagkraft zu besitzen. Vorwürfe gegen Web-Inhalte können sich theoretisch auf den Angriff auf die Menschenwürde (Art.1 GG) oder auf die Behinderung der freien Entfaltung der Persönlichkeit (Art.2 GG) berufen. Doch meistens nur im Zusammenhang mit anderen, konkreteren strafrechtlichen oder zivilrechtlichen Vorwürfen.

**Beachten Sie:** Im Link-Verzeichnis des Online-Angebots von SELFHTML bei *selfaktuell.teamone.de* aktuell finden Sie Verweise zu den relevanten Gesetzestexten.

### 25.6.4 Gesetze für »neue Medien« (in Deutschland)

In Deutschland gibt es mittlerweile Gesetze, die im Hinblick auf die neuen Realitäten des Internet geschaffen wurden. Diese Gesetze sollten Sie als Web-Autor kennen:

#### Informations- und Kommunikationsdienstegesetz (IuKDG)

Dieses Gesetz ist maßgeblich für elektronische Angebote »zur Information oder Kommunikation, soweit nicht die redaktionelle Gestaltung zur Meinungsbildung für die Allgemeinheit im Vordergrund steht (Datendienste, zum Beispiel Verkehrs-, Wetter-, Umwelt- und Börsendaten, Verbreitung von Informationen über Waren und Dienstleistungsangebote)« und für Angebote »von Waren und Dienstleistungen in elektronisch abrufbaren Datenbanken mit interaktivem Zugriff und unmittelbarer Bestellmöglichkeit« (Zitat aus Art.1,§ 2 – Geltungsbereich).

Obwohl durchaus diskutiert wird, ob etwa private Homepages in diesen Geltungsbereich fallen, ist das Gesetz aus derzeitiger Sicht durchaus für Homepage-Betreiber maßgeblich.



**Teledienstegesetz (TDG)**

Dieses Gesetz ist ein Auszug aus dem IuKDG und enthält den Artikel 1 daraus. Das Gesetz regelt aus inhaltlicher Anbietersicht die beiden folgenden wichtigen Punkte:

- Verantwortlichkeit (TDG, § 5)
- Anbieterkennzeichnung (TDG, § 6)

Wichtig ist dabei vor allem Absatz 2 aus § 5. Denn viele Web-Angebote »halten fremde Inhalte zur Benutzung bereit«, z.B. wenn Anwender selber Daten beitragen können, etwa bei Gästebüchern, Diskussionsforen usw. Als Anbieter solcher Services sind Sie für Inhalte ab dem Moment verantwortlich, wo Sie Kenntnis von den Inhalten erhalten.

**Teledienste-Datenschutzgesetz (TDDSG)**

Dieses Gesetz ist ein Auszug aus dem IuKDG und enthält den Artikel 2 daraus. Das Gesetz ist vor allem für Anbieter von Online-Shops wichtig, betrifft aber auch alle Anbieter, die aus anderen Gründen Benutzerdaten sammeln, z.B. um für ein Forum nur registrierte Benutzer zuzulassen. Folgende Punkte sind wichtig:

- Grundsätze für die Verarbeitung personenbezogener Daten (TDDSG, § 3)
- Datenschutzrechtliche Pflichten des Diensteanbieters (TDDSG, § 4)
- Bestandsdaten (TDDSG, § 5)
- Nutzungs- und Abrechnungsdaten (TDDSG, § 6)
- Auskunftsrecht des Nutzers (TDDSG, § 7)
- Datenschutzkontrolle (TDDSG, § 8)

Hervorzuheben ist dabei Absatz 4 des § 3, wo das Prinzip der so genannten »Datensparsamkeit« formuliert wird. Denn nicht wenige Anbieter von ganz normalen HTML-Formularen im Web haben bereits juristischen Ärger bekommen, weil sie mehr Pflichteingabefelder anboten, als es aus Gründen der Datensparsamkeit notwendig wäre. Die genannten Paragraphen enthalten auch verschiedene Regelung für die Speicherung personenbezogener Daten von Anwendern, die beispielsweise bei der Realisierung von CGI- und Datenbankanwendungen im Web zu berücksichtigen sind.

**Mediendienstestaatsvertrag (MdStV)**

Um einen Staatsvertrag und nicht um ein Bundesgesetz handelt es sich dabei deshalb, weil er die kulturelle Länderhoheit betrifft und deshalb ein von den Ländern einzeln unterzeichnetes Dokument darstellt. Die darin formulierten Paragraphen sind in vielen Fällen ähnlich wie Paragraphen aus TDG und TDDSG. Der MdStV enthält aber darüber hinaus noch einige weiterreichende Regelungen:

- Bei der Anbieterkennzeichnung (MdStV, § 6) wird für journalistisch-redaktionell gestaltete Angebote die so genannte Impressumpflicht formuliert.
- Es wird – im Gegensatz zum TDG – die Verpflichtung zu wahrheitsgemäßen Wiedergabe von Information zur Sorgfaltspflicht bei der Recherche postuliert (MdStV, § 7).

- Es werden ausdrücklich die strafrechtlichen Folgen im Falle von Volksverhetzung und jugendgefährdenden Inhalten betont (MdStV, § 8).
- Für Werbung wird festgelegt, dass diese als solche kenntlich gemacht werden muss (MdStV, §9).
- Das Prinzip der Gegendarstellung wird verlangt (MdStV, § 10). Dies verpflichtet den Anbieter, Widerspruch von Lesern/Anwendern gegen eine Tatsachenbehauptung der eigenen Redaktion zu veröffentlichen.

Interessant ist vor allem, inwiefern der MdStV für Web-Angebote anstelle von TDG und TDDSG relevant werden kann. Die Definitionen zum Geltungsbereich sind leider nicht ganz klar. Stark redaktionell orientierte Web-Angebote, die auch meinungsbildenden Charakter haben (E-Zines, Portale, viel besuchte und führende Angebote zu bestimmten Themen und Fachgebieten), sollten daher auch die erweiterten Bestimmungen des MdStV kennen und berücksichtigen.

**Beachten Sie:** Im Link-Verzeichnis des Online-Angebots von SELFHTML bei [selfaktuell.teamone.de](http://selfaktuell.teamone.de) aktuell finden Sie Verweise zu den relevanten Gesetzestexten.

### 25.6.5 Autoren-Aussage zur Link-Haftung

Die wohl prominenteste und für den Fortbestand des Web in der Tat gefährlichste rechtliche Problemzone ist die Frage der Link-Haftung. Obwohl Tim Berners Lee, der Gründervater des WWW, klar und deutlich Stellung dazu bezogen hat, was ein Link ist im Gegensatz etwa zu eingebetteten Fremdinhalten, tut sich die Rechtssprechung noch schwer damit. Immer noch wird vor Gericht die »Zeigerfunktion« eines Verweises mit dem Zueigenmachen fremder Inhalte verwechselt. Deshalb soll auch an dieser Stelle noch mal aus Sicht des Autors von SELFHTML (und damit aus Sicht »technisch informierter Fachkreise«) dargestellt werden, was ein Link (Synonym für Verweis) im Web ist.

Ein Verweis besteht aus zwei Teilen:

1. der Erwähnung des anderen Inhalts – sie ist im Verweistext formuliert, und
2. einer Meta-Information, nämlich zum Fundort des anderen Inhalts im Internet (»URI«).

Das gilt sowohl für Verweise innerhalb des eigenen Angebots als auch für Verweise zu fremden Angeboten. Verweise haben immer den gleichen Aufbau. In Bezug auf fremde Inhalte ist ein Verweis nichts anderes, als wenn ein fremder Inhalt vom Typ »Buch« erwähnt wird und dabei die Meta-Informationen für Erscheinungsdatum, Verlag und ISBN-Nummer hinzugefügt werden.

Den Zugang zum fremden Inhalt stellt aus technischer Sicht nicht der Anbieter des Verweises her, sondern der Anwender. Und zwar nicht zwangsläufig, sondern willentlich und nur dann, wenn er auf den Verweis klickt und damit die verlinkte Internet-Adresse aufruft. Das ist nichts anderes, wie wenn der Anwender sich entschließt, das erwähnte Buch zu erwerben, dazu Titel, Verlag und ISBN-Nummer auf einem Zettel notiert, sich in eine Buchhandlung oder Bibliothek begibt und das Buch dort erwirbt oder bestellt. Das Anklicken des Verweises führt nur zeitlich schneller zum Ziel, stellt aber qualitativ keinen Unterschied zum Gang in die Buchhandlung oder Bibliothek dar.

Wenn ein Angebot sich durch einen Link auf ein fremdes Angebot tatsächlich den fremden Inhalt zu eigen machen würde, dann hätte dies unter anderem folgende wichtige Konsequenzen:

- Eine große Suchmaschine, die das Suchen nach beliebigen Begriffen erlaubt und die Suchtreffer in Form von Links zu Fundorten auflistet, würde sich mehr oder weniger das gesamte Web zu eigen machen. Und selbst wenn man Suchmaschinen ausnehmen würde, weil sie weitgehend automatisiert arbeiten, würde die Aussage zumindest auf alle großen redaktionell gepflegten Web-Verzeichnisse zutreffen.
- Wer einen Link auf ein fremdes Angebot anbietet, das wieder auf ein anderes fremdes Angebot linkt, das wieder auf ein fremdes anderes Angebot linkt, würde sich im Sinne der infiniten teleologischen Reduktion alle weiteren Inhalte zu eigen machen.

### ***Persönliches Fazit***

Die einzig vernünftige Konsequenz für Inhaltsanbieter daraus wäre, keine Links mehr auf fremde Inhalte zu setzen, um die unkontrollierbare Verantwortung, die damit verbunden wäre, zu vermeiden. Vermutlich würden die Anwender dann auf ausländische Angebote ausweichen, deren Anbieter keine Angst beim Setzen von Links auf fremde Inhalte haben müssen. Web-Angebote des eigenen Landes würden dadurch einen gravierenden Wettbewerbsnachteil erleiden und vermutlich reihenweise aufgeben. Der Schaden für die Fortentwicklung neuer Medien im eigenen Land wäre unabsehbar.

# A Technische Ergänzungen

## A.1 HTTP-Statuscodes

### *Allgemeines zu HTTP-Statuscodes*

Wenn ein Web-Browser über das HTTP-Protokoll einen URI aufruft, erhält der Web-Server, der für die aufgerufene Domain oder IP-Adresse zuständig ist, die Anfrage. Er ermittelt die gewünschten Daten auf dem Server-Rechner und sendet diese – wieder über das HTTP-Protokoll – an den wartenden Web-Browser. Je nachdem, wie die Anforderung des Web-Browsers lautete und welches Ergebnis beim Ermitteln der Daten auf dem Server-Rechner herauskommt, übermittelt der Server in den HTTP-Kopfdaten der Antwort an den Browser eine Zahl. Diese Zahl stellt einen Statuscode dar und gibt Auskunft darüber, in welchem Verhältnis das, was der Server an Daten ausliefert, zu der gewünschten Anforderung steht. Auf diese Weise werden beispielsweise Fehlermeldungen übermittelt, aber auch Hinweise wie »die angeforderten Daten sind zugangsgeschützt«. Der Browser kann den Statuscode auswerten und entsprechend reagieren. Der Web-Server schreibt den HTTP-Statuscode der Anfrage außerdem in seine Log-Dateien.

HTTP-Statuscodes sind für Sie als Web-Anbieter an folgenden Stellen interessant:

- In den Log-Dateien des Web-Servers können Sie sich einen Eindruck davon machen, welche Fehler und Probleme in letzter Zeit auffällig oft auftauchten. Auch die meisten Statistiken zu Web-Projekten schlüsseln die Gesamtzahlen unter anderem nach HTTP-Statuscodes auf. Dabei ist es hilfreich, die einzelnen Statuscodes, ihre Bedeutung und mögliche Ursachen für ihr Vorkommen zu kennen.
- CGI-Scripts können selber HTTP-Header erzeugen. Dabei können Sie auch HTTP-Statuscodes notieren, die als Antwort an den aufrufenden Browser gesendet werden sollen. Der Web-Server reicht die Angaben dann an den Browser durch. Wie so etwas in einem CGI-Script mit Perl funktioniert, wird in Teil 2 des HTML & Web-Publishing Handbuchs im Abschnitt *Erweiterte Angaben für HTTP-Header und HTML-Dateikopf* (im Zusammenhang mit dem CGI-Modul von Perl) beschrieben.

Die HTTP-Statuscodes gehören zur Spezifikation des HTTP-Protokolls. Zuständig für die Spezifikation ist das W3-Konsortium. Die Spezifikation erfolgt in Form von Requests for Comments (RFCs). Über die folgende Seite des W3-Konsortiums können Sie alle relevanten und aktuelle Informationen rund um die Normierung des HTTP-Protokolls aufrufen ([www.w3.org/Protocols/](http://www.w3.org/Protocols/)).

**Übersicht der HTTP-Statuscodes**

| <i>Statuscode</i> | <i>Zugehörige Meldung</i>     | <i>Bedeutung</i>                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 100               | Continue                      | Fortfahren. Wird derzeit noch nicht verwendet.                                                                                                                                                                                                                                                                                                                                                                                 |
| 101               | Switching Protocols           | Protokoll wechseln. Wird derzeit noch nicht verwendet.                                                                                                                                                                                                                                                                                                                                                                         |
| 200               | OK                            | Der Server kann die angeforderten Daten wie gewünscht versenden. Dies ist der Normalfall, wenn keine Probleme auftauchen. Der größere Anteil aller Anfragen an den Web-Server sollte mit diesem Statuscode enden – ansonsten stimmt etwas nicht.                                                                                                                                                                               |
| 201               | Created                       | Ein Objekt (z.B. eine Datei oder ein Verzeichnis) wurde auf dem Server erfolgreich angelegt. Das kann vorkommen, wenn die Anfrage des Browsers an den Server mit einer der HTTP-Übertragungsmethoden post oder put erfolgte und eine Anweisung zum Erstellen des entsprechenden Objekts enthielt.                                                                                                                              |
| 202               | Accepted                      | Der Server hat die Anfrage des Browsers akzeptiert, liefert aber keine Daten als Antwort. Der Server schreibt die Daten statt sie zu senden in eine Datei und teilt in der Meldung mit, wo die Daten später zu finden sein werden. Diese Meldung sagt nichts darüber aus, ob der Server die Anfrage erfolgreich behandeln kann. Er hat sie einfach nur akzeptiert und die Abarbeitung auf einen späteren Zeitpunkt verschoben. |
| 203               | Non-Authoritative Information | Dieser Statuscode sollte von einem Server anstelle von 200 zurückgegeben werden, wenn es sich nicht um den Original-Server handelt, sondern beispielsweise um einen Proxy-Server. Der Web-Browser erfährt auf diese Weise, dass die Daten erfolgreich gesendet werden konnten, aber nicht vom Original-Server kommen und daher keine Garantie auf deren Aktualität besteht.                                                    |
| 204               | No Content                    | Der Server hat die Anfrage erhalten, sendet jedoch keine Daten zurück. Gut verwendbar ist dieser Statuscode bei Verwendung in CGI-Scripts, die zwar etwas auf dem Server erledigen, aber keinen neuen HTML-Code an den aufrufenden Browser senden wollen. Aus Sicht des Anwenders bleibt der alte Bildschirminhalt bestehen.                                                                                                   |
| 205               | Reset Content                 | Der angegebene Server existiert nicht bzw. der Server, der diese Antwort gibt, ist nicht der angefragte Server und kann den angefragten Server nicht finden. Die angeforderten Daten können deshalb nicht versendet werden.                                                                                                                                                                                                    |
| 206               | Partial Content               | Die angeforderten Daten werden in mehreren Portionen versendet (das hat aber nichts mit TCP/IP-Paketen zu tun, sondern passiert auf HTTP-Protokollebene). Mit Angaben zu content-length (z.B.: 1024) und content-range (z.B.: bytes 0-1023/1024) wird angegeben, wie viele Bytes von dem angeforderten Inhalt und welcher Teil der Gesamtdaten geliefert werden.                                                               |

| Statuscode | Zugehörige Meldung | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 300        | Multiple Choices   | Die angeforderten Daten sind unter mehreren verschiedenen URIs vorhanden (Mirror). Anstelle der Daten werden die verfügbaren URIs als Liste übertragen. Der Web-Browser kann den Anwender anschließend in einem Dialog einen URI auswählen lassen.                                                                                                                                                                                                                                              |
| 301        | Moved Permanently  | Die angeforderten Daten befinden sich nicht mehr unter dem URI, sie wurden dauerhaft auf eine andere Adresse verschoben. In der Statusmeldung wird angegeben, unter welchem URI sich die Daten jetzt befinden. Ein Web-Browser, der diese Antwort vom Server erhält, kann beispielsweise gleich die neue Adresse anfordern.                                                                                                                                                                     |
| 302        | Moved Temporarily  | Die angeforderten Daten wurden vorübergehend zu einem anderen URI verschoben. In der Statusmeldung wird angegeben, unter welcher Adresse sich die Daten derzeit befinden. Ein Web-Browser, der diese Antwort erhält, kann beispielsweise gleich die temporär gültige Adresse anfordern.                                                                                                                                                                                                         |
| 303        | See Other          | Die angeforderten Daten sind unter einem angegebenen URI verfügbar und sollten von dort mit Hilfe der get-Methode angefordert werden. Dieser Statuscode ist für CGI-Scripts gedacht, die mit der post-Methode aufgerufen wurden und den Browser auf eine andere Ressource lenken wollen, die mit der get-Methode angefordert werden soll.                                                                                                                                                       |
| 304        | Not Modified       | Die angeforderten Daten haben sich seit dem angegebenen Zeitpunkt nicht geändert und werden deshalb nicht gesendet. Dieser Statuscode ist neben dem Code 200 einer der häufigsten in der Praxis. Er wird verursacht durch Web-Browser, die aufgrund ihrer Cache-Einstellungen Daten erst wieder nach einer bestimmten Zeit vom Original-Server laden. Davor fragen sie nur mit dem Zeitpunkt, zu dem die Daten zuletzt geladen wurden, an, ob die Daten auf dem Server seitdem geändert wurden. |
| 305        | Use Proxy          | Die angeforderten Daten sollen statt von diesem Server von dem in der Statusmeldung angegebenen Proxy-Server angefordert werden.                                                                                                                                                                                                                                                                                                                                                                |
| 306        | [Unused]           | Reserviert. Wird derzeit aber nicht verwendet.                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 307        | Temporary Redirect | Wie Statuscode 302. Gedacht für Fehlreaktionen einiger Browser auf 302.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 400        | Bad Request        | Die Anfrage enthält Syntaxfehler. Der Server kann die Anfrage deshalb nicht bearbeiten. Das kann beispielsweise vorkommen, wenn die Anfrage dadurch zustande kam, dass ein Anwender versuchte, einen URI händisch in die Adresszeile des Browsers einzugeben und dabei ungültige Zeichen verwendete.                                                                                                                                                                                            |

| <i>Statuscode</i> | <i>Zugehörige Meldung</i> | <i>Bedeutung</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 401               | Unauthorized              | Die angeforderten Daten sind zugangsgeschützt. Der Server kann die Daten nur senden, wenn eine gültige Zugangsken-<br>nung, bestehend aus Benutzername und Passwort, bei der<br>Anfrage mitgesendet wird. Das passiert in der Praxis immer<br>dann, wenn eine Adresse aufgerufen wird, die z.B.<br>durch htaccess zugangsgeschützt ist. Der Web-Browser zeigt<br>dann, nachdem er diesen Statuscode erhalten hat, einen Dia-<br>log zum Eingeben von Benutzername und Kennwort an. Mit<br>den eingegebenen Daten startet er dann eine neue Anfrage an<br>den Server.                                     |
| 402               | Payment Required          | Die angeforderten Daten sind kostenpflichtig. Der Server<br>kann die Daten nur senden, wenn eine Bestätigung der Zah-<br>lung für die Daten bei der Anfrage mitgesendet wird. Derzeit<br>wird dies aufgrund fehlender einheitlicher technischer<br>Grundlagen für Micropayment noch nicht verwendet.                                                                                                                                                                                                                                                                                                     |
| 403               | Forbidden                 | Die angeforderten Daten sind zugangsgeschützt. Die ange-<br>gebenen Daten, mit denen der Zugang erlaubt werden soll, sind<br>ungültig. Das kann z.B. vorkommen, wenn zuvor der Status-<br>code 401 zurückgeliefert worden war und der Browser nun die<br>nächste Anfrage mit den Zugangsdaten gestartet hat, die er<br>vom Anwender im Dialog abgefragt hat, und diese Daten aber<br>ungültig sind. Manche Web-Browser wiederholen den Dialog<br>zum Eingeben der Zugangsdaten dann noch zweimal, und<br>nach der dritten Falscheingabe wird dem Anwender die Feh-<br>lermeldung »Forbidden« ausgegeben. |
| 404               | Not Found                 | Der angeforderte URI existiert nicht. Dies ist neben den Sta-<br>tuscodes 200 und 304 einer der häufigsten Fälle in der Praxis.<br>Er tritt immer dann ein, wenn ein Verweis auf eine nicht oder<br>nicht mehr existierende Adresse auf dem Server führt oder<br>wenn der Anwender versucht hat, eine Adresse auf dem Server<br>durch händisches Eintippen in der Adresszeile des Browsers<br>aufzurufen, und diese Adresse aber nicht existiert.                                                                                                                                                        |
| 405               | Method Not Allowed        | Die angegebene Übertragungsmethode ist auf dem Server<br>nicht erlaubt. Die Daten werden deshalb nicht übertragen.<br>Das kann beispielsweise vorkommen, wenn in der Konfigura-<br>tion des Web-Servers außer der get-Methode keine weitere<br>Methode erlaubt ist, ein HTML-Formular aber einen CGI-<br>Aufruf mit der post-Methode enthält.                                                                                                                                                                                                                                                            |
| 406               | Not Acceptable            | Die Anfrage ist in dieser Form nicht akzeptabel. Die Daten<br>werden deshalb nicht übertragen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

| Statuscode | Zugehörige Meldung            | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 407        | Proxy Authentication Required | Der anfragende Client ist ein Proxy-Server. Die Daten werden an diesen Server nur übertragen, wenn er sich als gültiger Proxy-Server ausweist. Dieser Statuscode findet derzeit noch keine Verwendung. Damit soll auf die Dauer ein ähnliches Handling wie mit dem Statuscode 401 etabliert werden, jedoch nicht für anfragende Browser, sondern für anfragende Proxy-Server. Auf diese Weise könnte es Web-Anbietern möglich werden, in der Serverkonfiguration unerwünschte Proxy-Server vom Zwischenspeichern der eigenen Daten auszusperren. |
| 408        | Request Timeout               | Der Server hat eine erwartete Anfrage nicht innerhalb des dafür festgelegten Maximalzeitraums erhalten. Die Verbindung zum anfragenden Browser wird deshalb abgebaut. Angeforderte Daten werden nicht übertragen.                                                                                                                                                                                                                                                                                                                                |
| 409        | Conflict                      | Der Server kann die angeforderten Daten nicht senden, weil ein Konflikt mit einem anderen Prozess aufgetaucht ist. Das kann beispielsweise passieren, wenn ein anderer Prozess eine angeforderte Datei gerade mit einem exklusiven File-Locking (keinerlei Dateizugriff für andere Prozesse nicht erlaubt) versehen hat.                                                                                                                                                                                                                         |
| 410        | Gone                          | Die angeforderten Daten wurden zu einem anderen URI verschoben. Dem Server ist aber nicht bekannt, wohin. Deshalb kann er sie nicht senden – andernfalls wäre ein Statuscode 301 oder 302 gesendet worden.                                                                                                                                                                                                                                                                                                                                       |
| 411        | Length Required               | Die Daten werden nicht gesendet. Sie können nur gesendet werden, wenn die Anfrage eine Angabe zu content-length enthält. Der Browser kann versuchen, die Anfrage neu zu formulieren und dabei die Länge der an den Server gesendeten Anfragedaten mit zu übermitteln.                                                                                                                                                                                                                                                                            |
| 412        | Precondition Failed           | Eine oder mehrere Bedingungen, die bei der Anfrage gestellt wurden, treffen nicht zu. Die angeforderten Daten werden deshalb nicht übertragen.                                                                                                                                                                                                                                                                                                                                                                                                   |
| 413        | Request Entity Too Large      | Der Server kann die Anfrage nicht bearbeiten, weil diese zu viele Zeichen enthält. Die angeforderten Daten werden deshalb nicht übertragen.                                                                                                                                                                                                                                                                                                                                                                                                      |
| 414        | Request-URL Too Long          | Der Server kann die Anfrage nicht bearbeiten, weil die angeforderte Adresse zu viele Zeichen enthält. Die angeforderten Daten werden deshalb nicht übertragen.                                                                                                                                                                                                                                                                                                                                                                                   |
| 415        | Unsupported Media Type        | Der Server kann die Anfrage nicht bearbeiten, weil er keinen Mime-Type für den angeforderten Datentyp kennt. Die angeforderten Daten werden deshalb nicht übertragen.                                                                                                                                                                                                                                                                                                                                                                            |



| Statuscode | Zugehörige Meldung              | Bedeutung                                                                                                                                                                                                                                                                                                                                                                           |
|------------|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 416        | Requested Range Not Satisfiable | Die Anfrage enthält Angaben, welcher Byte-Bereich von dem angeforderten URI übertragen werden soll. Sowohl der Anfangswert als auch der Endwert des angegebenen Bereichs liegen außerhalb des verfügbaren Byte-Bereichs, z.B. wenn ein Byte-Bereich von 1000 bis 2000 angegeben wird, die Ressource aber nur 500 Byte hat. Die angeforderten Daten werden deshalb nicht übertragen. |
| 417        | Expectation Failed              | Die Anfrage enthält im expect-Feld bestimmte Wünsche, die der Server nicht erfüllen kann. Die angeforderten Daten werden deshalb nicht übertragen.                                                                                                                                                                                                                                  |
| 500        | Internal Server Error           | Der Server kann die angeforderten Daten nicht senden, weil auf dem Server ein Fehler aufgetreten ist. Beispielsweise konnte das aufgerufene CGI-Script nicht gestartet werden.                                                                                                                                                                                                      |
| 501        | Not Implemented                 | Die Anfrage enthält Anforderungen, die der Server nicht bearbeiten kann, weil die Voraussetzungen dazu nicht implementiert sind. Die angeforderten Daten können deshalb nicht gesendet werden.                                                                                                                                                                                      |
| 502        | Bad Gateway                     | Zum Bearbeiten der Anfrage musste der Server einen anderen Server aufrufen, erhielt dabei jedoch eine Fehlermeldung. Die angeforderten Daten können deshalb nicht gesendet werden.                                                                                                                                                                                                  |
| 503        | Service Unavailable             | Der Server kann die Anfrage wegen Überlastung nicht bearbeiten. Die angeforderten Daten können deshalb nicht gesendet werden. In der Statusmeldung kann stehen, wann die Anfrage frühestens wieder bearbeitet werden kann. Im Gegensatz zum Statuscode 202 verarbeitet der Server die Daten nicht, sobald er wieder Kapazitäten hat.                                                |
| 504        | Gateway Timeout                 | Zum Bearbeiten der Anfrage musste der Server einen anderen Server aufrufen, erhielt dabei jedoch nach einem festgelegten Maximalzeitraum keine Antwort. Die angeforderten Daten können deshalb nicht gesendet werden.                                                                                                                                                               |
| 505        | HTTP Version Not Supported      | Der Server unterstützt die im HTTP-Header der Anfrage angegebene HTTP-Version nicht. Die angeforderten Daten werden deshalb nicht gesendet.                                                                                                                                                                                                                                         |

## A.2 Mime-Typen

### *Allgemeines zu Mime-Typen*

Mime steht für Multipurpose Internet Mail Extensions. Aus dem Namen geht hervor, dass das, was da spezifiziert wird, ursprünglich für E-Mails gedacht war – und zwar für E-Mails mit Attachments (englisch für *Anhang*). Solche so genannten Multipart-Mails enthalten die gesamten zu übertragenden Daten in einer Datei. Innerhalb der Datei musste eine Konvention gefunden werden, wie die einzelnen Teile (z.B. Text der Mail und angehängte ZIP-Datei) voneinander zu trennen seien. Dabei wurde auch ein Schema entwickelt, das der interpretierenden Software

mitteilt, um welchen Datentyp es sich bei dem jeweils nächsten Teil der Mail handelt. Das Schema erwies sich nicht nur für E-Mails als nützlich. Fast immer, wenn entfernte Programme (z.B. Web-Browser und Web-Server) wegen einer bevorstehenden Datenübertragung miteinander kommunizieren, geht es auch um die Art der zu übertragenden Daten. Dabei hat sich im gesamten Internet das Schema der Mime-Typen durchgesetzt. Auch im Web stößt man an verschiedenen Stellen auf Mime-Typen:

- Verschiedene HTML-Elemente haben Attribute, die als Wertzuweisung die Angabe eines Mime-Typen erwarten, nämlich `a (type=)`, `form (accept= und enctype=)`, `input (accept=)`, `link (type=)`, `object (codetype= und type=)`, `param (type=)`, `script (type=)` und `style (type=)`
- Wenn ein CGI-Script an den aufrufenden Browser einen HTTP-Header sendet, muss es den Mime-Type der nachfolgenden Daten senden.
- In JavaScript gibt es ein eigenes Objekt `MimeTypes`, das es erlaubt, in JavaScript die verfügbaren Mime-Typen des Browsers zu ermitteln.

Sowohl jeder Web-Browser als auch jeder Web-Server führen eine Liste mit ihnen bekannten Mime-Typen. Bei der Kommunikation müssen sie sich darauf einigen, ob der Empfänger den Mime-Type akzeptiert, den der Sender senden will. Moderne Browser akzeptieren zwar in der Regel jeden Mime-Type und bieten dem Anwender einfach an, falls sie den Mime-Type nicht kennen, die zu empfangenden Daten als Download-Datei abzuspeichern. Web-Server sind dagegen meist empfindlicher. Mime-Typen, die sie nicht kennen, verarbeiten sie nicht. Gerade wenn auf dem Server-Rechner nicht alltägliche Dateiformate bereitgestellt werden, ist es wichtig, den Mime-Type dafür in der Konfiguration des Web-Servers zu notieren.

Bei Standard-Dateiformaten sollten Sie unbedingt die Mime-Type-Angaben verwenden, die dafür etabliert sind. Die Übersicht auf dieser Seite weiter unten listet viele bekannte Mime-Typen auf. Wenn Sie trotz Recherchen, zum Beispiel im IANA-Verzeichnis der Media-Typen ([www.isi.edu/in-notes/iana/assignments/media-types/](http://www.isi.edu/in-notes/iana/assignments/media-types/)) keinerlei Anhaltspunkte dafür finden, ob es zu einem Dateiformat einen bereits etablierten Mime-Type gibt, können Sie selber einen festlegen. Gleiches gilt für völlig eigene Dateiformate.

Dazu müssen Sie das Schema der Mime-Typen kennen. Ein Mime-Type besteht aus zwei Teilen: der Angabe eines Medientyps und der Angabe eines Subtyps. Beide Angaben werden durch einfachen Schrägstrich voneinander getrennt. Beispiele: `text/html`, `image/gif`.

Bei den Medientypen gibt es folgende:

| Medientyp                | Bedeutung                                                   |
|--------------------------|-------------------------------------------------------------|
| <code>text</code>        | für Textdateien                                             |
| <code>image</code>       | für Grafikdateien                                           |
| <code>video</code>       | für Videodateien                                            |
| <code>audio</code>       | für Sound-Dateien                                           |
| <code>application</code> | für Dateien, die an ein bestimmtes Programm gebunden sind   |
| <code>multipart</code>   | für mehrteilige Daten                                       |
| <code>message</code>     | für Nachrichten                                             |
| <code>model</code>       | für Dateien, die mehrdimensionale Strukturen repräsentieren |

Subtypen für servereigene Dateiformate, d.h. Dateitypen, die auf dem Server ausgeführt werden können, werden meist mit *x*- eingeleitet. Das Schema der Mime-Types wird in den Requests for Comments (RFCs) mit den Nummern 2045, 2046 und 2077 festgelegt.

### Übersicht von Mime-Typen

Die Auswahl ist alphabetisch sortiert und erhebt keinen Anspruch auf Vollständigkeit.

| <i>Mime-Type</i>                | <i>Dateiendung(en)</i>             | <i>Bedeutung</i>                      |
|---------------------------------|------------------------------------|---------------------------------------|
| application/acad                | *.dwg                              | AutoCAD-Dateien (nach NCSA)           |
| application/applefile           |                                    | AppleFile-Dateien                     |
| application/astound             | *.asd *.asn                        | Astound-Dateien                       |
| application/dsptype             | *.tsp                              | TSP-Dateien                           |
| application/dxf                 | *.dxf                              | AutoCAD-Dateien (nach CERN)           |
| application/futuresplash        | *.spl                              | Flash Futuresplash-Dateien            |
| application/gzip                | *.gz                               | GNU Zip-Dateien                       |
| application/listenup            | *.ptlk                             | Listenup-Dateien                      |
| application/mac-binhex40        | *.hqx                              | Macintosh Binär-Dateien               |
| application/mbedlet             | *.mbd                              | Mbedlet-Dateien                       |
| application/mif                 | *.mif                              | FrameMaker Interchange Format-Dateien |
| application/msexcel             | *.xls *.xla                        | Microsoft Excel-Dateien               |
| application/mshelp              | *.hlp *.chm                        | Microsoft Windows Hilfe-Dateien       |
| application/mspowerpoint        | *.ppt *.ppz *.pps *.pot            | Microsoft Powerpoint-Dateien          |
| application/msword              | *.doc *.dot                        | Microsoft Word-Dateien                |
| application/octet-stream        | *.bin *.exe *.com *.dll<br>*.class | Ausführbare Dateien                   |
| application/oda                 | *.oda                              | Oda-Dateien                           |
| application/pdf                 | *.pdf                              | Adobe PDF-Dateien                     |
| application/postscript          | *.ai *.eps *.ps                    | Adobe Postscript-Dateien              |
| application/rtc                 | *.rtc                              | RTC-Dateien                           |
| application/rtf                 | *.rtf                              | Microsoft RTF-Dateien                 |
| application/studiom             | *.smp                              | Studiom-Dateien                       |
| application/toolbook            | *.tbk                              | Toolbook-Dateien                      |
| application/vocaltec-media-desc | *.vmd                              | Vocaltec Mediadesc-Dateien            |
| application/vocaltec-media-file | *.vmf                              | Vocaltec Media-Dateien                |
| application/x-bcpio             | *.bcpio                            | BCPIO-Dateien                         |
| application/x-compress          | *.z                                | Z-Dateien                             |
| application/x-cpio              | *.cpio                             | CPIO-Dateien                          |
| application/x-csh               | *.csh                              | C-Shellscript-Dateien                 |
| application/x-director          | *.dcr *.dir *.dxr                  | X-Director-Dateien                    |
| application/x-dvi               | *.dvi                              | DVI-Dateien                           |
| application/x-envoy             | *.evy                              | Envoy-Dateien                         |
| application/x-gtar              | *.gtar                             | GNU tar-Archiv-Dateien                |

| <i>Mime-Typ</i>                     | <i>Dateiendung(en)</i> | <i>Bedeutung</i>                      |
|-------------------------------------|------------------------|---------------------------------------|
| application/x-hdf-*.hdf-HDF-Dateien |                        |                                       |
| application/x-httpd-php             | *.php *.phtml          | PHP-Dateien                           |
| application/x-javascript            | *.js                   | serverseitige JavaScript-Dateien      |
| application/x-latex                 | *.latex                | Latex-Quelldateien                    |
| application/x-macbinary             | *.bin                  | Macintosh Binärdateien                |
| application/x-mif                   | *.mif                  | FrameMaker Interchange Format-Dateien |
| application/x-netcdf                | *.nc *.cdf             | Unidata CDF-Dateien                   |
| application/x-nschat                | *.nsc                  | NS Chat-Dateien                       |
| application/x-sh                    | *.sh                   | Bourne Shellscrip-Dateien             |
| application/x-shar                  | *.shar                 | Shell-Archiv-Dateien                  |
| application/x-shockwave-flash       | *.swf *.cab            | Flash Shockwave-Dateien               |
| application/x-sprite                | *.spr *.sprite         | Sprite-Dateien                        |
| application/x-stuffit               | *.sit                  | Stuffit-Dateien                       |
| application/x-supercard             | *.sca                  | Supercard-Dateien                     |
| application/x-sv4cpio               | *.sv4cpio              | CPIO-Dateien                          |
| application/x-sv4crc                | *.sv4crc               | CPIO-Dateien mit CRC                  |
| application/x-tar                   | *.tar                  | tar-Archivdateien                     |
| application/x-tcl                   | *.tcl                  | TCL Scriptdateien                     |
| application/x-tex                   | *.tex                  | TEX-Dateien                           |
| application/x-texinfo               | *.texinfo *.texi       | TEXinfo-Dateien                       |
| application/x-troff                 | *.t *.tr *.roff        | TROFF-Dateien (Unix)                  |
| application/x-troff-man             | *.man *.troff          | TROFF-Dateien mit MAN-Makros (Unix)   |
| application/x-troff-me              | *.me *.troff           | TROFF-Dateien mit ME-Makros (Unix)    |
| application/x-troff-ms              | *.me *.troff           | TROFF-Dateien mit MS-Makros (Unix)    |
| application/x-ustar                 | *.ustar                | tar-Archivdateien (Posix)             |
| application/x-wais-source           | *.src                  | WAIS Quelldateien                     |
| application/x-www-form-urlencoded   |                        | HTML-Formulardaten an CGI             |
| application/zip                     | *.zip                  | ZIP-Archivdateien                     |
| audio/basic                         | *.au *.snd             | Sound-Dateien                         |
| audio/echospeech                    | *.es                   | Echospeed-Dateien                     |
| audio/tsplayer                      | *.tsi                  | TS-Player-Dateien                     |
| audio/voxware                       | *.vox                  | Vox-Dateien                           |
| audio/x-aiff                        | *.aif *.aiff *.aifc    | AIFF-Sound-Dateien                    |
| audio/x-dspeek                      | *.dus *.cht            | Sprachdateien                         |
| audio/x-midi                        | *.mid *.midi           | MIDI-Dateien                          |
| audio/x-mpeg                        | *.mp2                  | MPEG-Dateien                          |
| audio/x-pn-realaudio                | *.ram *.ra             | RealAudio-Dateien                     |
| audio/x-pn-realaudio-plugin         | *.rpm                  | RealAudio-Plug-In-Dateien             |
| audio/x-qt-stream                   | *.stream               | QuickTime-Dateien                     |
| audio/x-wav                         | *.wav                  | Wav-Dateien                           |
| drawing/x-dwf                       | *.dwf                  | Drawing-Dateien                       |
| image/cis-cod                       | *.cod                  | CIS-Cod-Dateien                       |

| <i>Mime-Typ</i>             | <i>Dateiendung(en)</i>      | <i>Bedeutung</i>                                          |
|-----------------------------|-----------------------------|-----------------------------------------------------------|
| image/cmu-raster            | <i>*.ras</i>                | CMU-Raster-Dateien                                        |
| image/fif                   | <i>*.fif</i>                | FIF-Dateien                                               |
| image/gif                   | <i>*.gif</i>                | GIF-Dateien                                               |
| image/ief                   | <i>*.ief</i>                | IEF-Dateien                                               |
| image/jpeg                  | <i>*.jpeg *.jpg *.jpe</i>   | JPEG-Dateien                                              |
| image/tiff                  | <i>*.tiff *.tif</i>         | TIFF-Dateien                                              |
| image/vasa                  | <i>*.mcf</i>                | Vasa-Dateien                                              |
| image/vnd.wap.wbmp          | <i>*.wbmp</i>               | Bitmap-Dateien (WAP)                                      |
| image/x-freehand            | <i>*.fh4 *.fh5 *.fhc</i>    | Freehand-Dateien                                          |
| image/x-portable-anymap     | <i>*.pnm</i>                | PBM Anymap-Dateien                                        |
| image/x-portable-bitmap     | <i>*.pbm</i>                | PBM Bitmap-Dateien                                        |
| image/x-portable-graymap    | <i>*.pgm</i>                | PBM Graymap-Dateien                                       |
| image/x-portable-pixmap     | <i>*.ppm</i>                | PBM Pixmap-Dateien                                        |
| image/x-rgb                 | <i>*.rgb</i>                | RGB-Dateien                                               |
| image/x-windowdump          | <i>*.xwd</i>                | X-Windows Dump                                            |
| image/x-xbitmap             | <i>*.xbm</i>                | XBM-Dateien                                               |
| image/x-xpixmap             | <i>*.xpm</i>                | XPM-Dateien                                               |
| message/external-body       |                             | Nachricht mit externem Inhalt                             |
| message/http                |                             | HTTP-Headernachricht                                      |
| message/news                |                             | Newsgroup-Nachricht                                       |
| message/partial             |                             | Nachricht mit Teilinhalt                                  |
| message/rfc822              |                             | Nachricht nach RFC 1822                                   |
| model/vrml                  | <i>*.wrl</i>                | Visualisierung virtueller Welten                          |
| multipart/alternative       |                             | mehrteilige Daten gemischt                                |
| multipart/byteranges        |                             | mehrteilige Daten mit Byte-Angaben                        |
| multipart/digest            |                             | mehrteilige Daten / Auswahl                               |
| multipart/encrypted         |                             | mehrteilige Daten verschlüsselt                           |
| multipart/form-data         |                             | mehrteilige Daten aus HTML-Formular<br>(z.B. File-Upload) |
| multipart/mixed             |                             | mehrteilige Daten gemischt                                |
| multipart/parallel          |                             | mehrteilige Daten parallel                                |
| multipart/related           |                             | mehrteilige Daten / verbunden                             |
| multipart/report            |                             | mehrteilige Daten / Bericht                               |
| multipart/signed            |                             | mehrteilige Daten / bezeichnet                            |
| multipart/voice-message     |                             | mehrteilige Daten / Sprachnachricht                       |
| text/comma-separated-values | <i>*.csv</i>                | kommaseparierte Datendateien                              |
| text/css                    | <i>*.css</i>                | CSS Stylesheet-Dateien                                    |
| text/html                   | <i>*.htm *.html *.shtml</i> | HTML-Dateien                                              |
| text/javascript             | <i>*.js</i>                 | JavaScript-Dateien                                        |
| text/plain                  | <i>*.txt</i>                | reine Textdateien                                         |
| text/richtext               | <i>*.rtx</i>                | Richtext-Dateien                                          |
| text/rtf                    | <i>*.rtf</i>                | Microsoft RTF-Dateien                                     |
| text/tab-separated-values   | <i>*.tsv</i>                | tabulatorseparierte Datendateien                          |
| text/vnd.wap.wml            | <i>*.wml</i>                | WML-Dateien (WAP)                                         |

| Mime-Typ                        | Dateiendung(en)              | Bedeutung                   |
|---------------------------------|------------------------------|-----------------------------|
| application/vnd.wap.wmlc        | *.wmlc                       | WMLC-Dateien (WAP)          |
| text/vnd.wap.wmlscript          | *.wmls                       | WML-Scriptdateien (WAP)     |
| application/vnd.wap.wmlscriptc  | *.wmlsc                      | WML-Script-C-Dateien (WAP)  |
| text/xml-external-parsed-entity |                              | extern geparste XML-Dateien |
| text/x-setext                   | *.etx                        | SeText-Dateien              |
| text/x-sgml                     | *.sgm *.sgml                 | SGML-Dateien                |
| text/x-speech                   | *.talk *.spc                 | Speech-Dateien              |
| video/mpeg                      | *.mpeg *.mpg *.mpe           | MPEG-Dateien                |
| video/quicktime                 | *.qt *.mov                   | Quicktime-Dateien           |
| video/vnd.vivo                  | *viv *.vivo                  | Vivo-Dateien                |
| video/x-msvideo                 | *.avi                        | Microsoft AVI-Dateien       |
| video/x-sgi-movie               | *.movie                      | Movie-Dateien               |
| workbook/formulaone             | *.vts *.vtts                 | FormulaOne-Dateien          |
| x-world/x-3dmf                  | *.3dmf *.3dm *.qd3d<br>*.qd3 | 3DMF-Dateien                |
| x-world/x-vrml                  | *.wrl                        | VRML-Dateien                |

## A.3 Sprachenkürzel

### Sprachenkürzel nach RFC 1766

Die in den W3-Spezifikationen immer wieder erwähnte »Angabe von Sprachenkürzeln« wird in der RFC 1766 definiert. Diese Sprachenkürzel basieren auf den Normen ISO 639-1 zur Kurzbezeichnung von Sprachen und ISO 3166-1 zur Kurzbezeichnung von Länder. Diese von den beiden ISO-Normen vordefinierten Kürzel erlauben solche Angaben wie *de* für Deutsch, *en* für Englisch oder *en-US* für US-Englisch. Die RFC 1766 erlaubt darüber hinaus, feinere Abstufungen für schriftkulturelle, dialektale oder andere lokale Ausprägungen der gleichen Sprache registrieren zu lassen, so z.B. die beiden norwegischen Schriftsprachenvarianten, notiert durch *no-nyn* und *no-bok* (wobei diese – in Langform *no-nynorsk* und *no-bokmaal* in der RFC 1766 beispielhaft aufgezählten – Kürzel zwischenzeitlich veraltet sind, da es mittlerweile die zweibuchstabigen Kürzel *nn* und *nb* gibt). Für Sprachen, die in mehreren Ländern oder von bestimmten Gruppen gesprochen werden, und bei denen die Länderzuordnung deshalb problematisch ist, wird das einbuchstabile Startkürzel *i* reserviert. Die Reservierung erfolgt durch die Internet-Behörde IANA ([www.iana.org/](http://www.iana.org/)). So werden Bezeichnungen wie *i-mingo* (eine Sprache der Irokesen) möglich. Bei der Reservierung beweist IANA Humor: Auch das Sprachkürzel *x-klingon* (für »Klingonisch« – aus dem Kultserie »Startrek«) wurde registriert. Außerdem wird ein Startkürzel *x* für den privaten Gebrauch reserviert, wodurch Sprachenkürzel wie *x-selfhtml* (für »SELFHTMLisch«) bezeichnet werden können.

Mittlerweile gibt es die RFC 3066, nach der auch dreibuchstabile Sprachenkürzel nach der Norm ISO 639-2 erlaubt sind, etwa *nds* für Niederdeutsch. Die HTML- und CSS-Spezifikationen des W3-Konsortiums beziehen sich an den relevanten Stellen jedoch noch auf die ältere RFC 1766. Für XML gilt die RFC 3066. Ebenso gibt es eine neuere Norm ISO 3166-2, die mehrbuchstabile

Ländercodes enthält. Maßgeblich für Internet-Belange sind aber in den meisten Fällen noch die älteren, zweibuchstabigen Kürzel.

Für Web-Autoren haben die Sprachenkürzel an folgenden Stellen Bedeutung:

- Verschiedene HTML-Elemente haben Attribute, die als Wertzuweisung die Angabe einer Sprache erwarten. So gibt es das Universalattribut `lang=`, und ebenfalls erwartet wird eine Sprachangabe bei den Elementen (Attributen) `a (hreflang=)` und `link (hreflang=)`.
- In JavaScript haben die Sprachenkürzel beispielsweise bei der Eigenschaft `navigator.language` Bedeutung.
- Im Zusammenhang mit der CGI-Schnittstelle gibt es die Umgebungsvariable `HTTP_ACCEPT_LANGUAGE`, in der solche Sprachenkürzel verwendet werden.

Länderkürzel nach ISO 3166 werden auch noch an anderen Stellen eingesetzt, so etwa bei den landesspezifischen Top-Level-Domains, wie sie beispielsweise in den Statistiken zu einem Web-Projekt auftauchen können.

### **Übersicht zu Sprachenkürzeln nach ISO 639-1**

Die folgenden Sprachenkürzel können Sie auch überall dort verwenden, wo Angaben nach RFC 1766 gefordert sind. In den meisten Fällen reichen die Original-Sprachenkürzel nach ISO 639-1 zur Bezeichnung der Sprache auch vollkommen aus. Die Tabelle ist nach Kürzeln alphabetisch sortiert und erhebt keinen Anspruch auf aktuelle Vollständigkeit. Streichungen, Ergänzungen, Neueinträge von Sprachenkürzeln notiert die Library of Congress (dokumentiert auf der Seite [ISO 639-2/RA Change Notice, \*lweb.loc.gov/standards/iso639-2/codechanges.html\*](http://iso639-2/RA/ChangeNotice)).

| <i>Kürzel</i> | <i>Bedeutung</i>     | <i>Kürzel</i> | <i>Bedeutung</i>  | <i>Kürzel</i> | <i>Bedeutung</i> |
|---------------|----------------------|---------------|-------------------|---------------|------------------|
| aa            | Afar                 | ab            | Abchasisch        | af            | Afrikaans        |
| am            | Amharisch            | ar            | Arabisch          | as            | Assamesisch      |
| ay            | Aymara               | az            | Aserbaidshisch    | ba            | Baschkirisch     |
| be            | Belorussisch         | bg            | Bulgarisch        | bh            | Biharisch        |
| bi            | Bislamisch           | bn            | Bengalisch        | bo            | Tibetanisch      |
| br            | Bretonisch           | ca            | Katalanisch       | co            | Korsisch         |
| cs            | Tschechisch          | cy            | Walisisch         | da            | Dänisch          |
| de            | Deutsch              | dz            | Dzongkha, Bhutani | el            | Griechisch       |
| en            | Englisch             | eo            | Esperanto         | es            | Spanisch         |
| et            | Estnisch             | eu            | Baskisch          | fa            | Persisch         |
| fi            | Finnisch             | fj            | Fiji              | fo            | Faröisch         |
| fr            | Französisch          | fy            | Friesisch         | ga            | Irish            |
| gd            | Schottisches Gälisch | gl            | Galizisch         | gn            | Guarani          |
| gu            | Gujaratisch          | ha            | Hausa             | he            | Hebräisch        |
| hi            | Hindi                | hr            | Kroatisch         | hu            | Ungarisch        |
| hy            | Armenisch            | ia            | Interlingua       | id            | Indonesisch      |
| ie            | Interlingue          | ik            | Inupiak           | is            | Isländisch       |

| <i>Kürzel</i> | <i>Bedeutung</i> | <i>Kürzel</i> | <i>Bedeutung</i>            | <i>Kürzel</i> | <i>Bedeutung</i>              |
|---------------|------------------|---------------|-----------------------------|---------------|-------------------------------|
| it            | Italienisch      | iu            | Inuktitut (Eskimo)          | iw            | Hebräisch (veraltet, nun: he) |
| ja            | Japanisch        | ji            | Jiddish (veraltet, nun: yi) | jv            | Javanisch                     |
| ka            | Georgisch        | kk            | Kasachisch                  | kl            | Kalaallisut (Grönländisch)    |
| km            | Kambodschanisch  | kn            | Kannada                     | ko            | Koreanisch                    |
| ks            | Kaschmirisch     | ku            | Kurdisch                    | ky            | Kirgisisch                    |
| la            | Lateinisch       | ln            | Lingala                     | lo            | Laotisch                      |
| lt            | Litauisch        | lv            | Lettisch                    | mg            | Malagasisch                   |
| mi            | Maorisch         | mk            | Mazedonisch                 | ml            | Malajalam                     |
| mn            | Mongolisch       | mo            | Moldavisch                  | mr            | Marathi                       |
| ms            | Malaysisch       | mt            | Maltesisch                  | my            | Burmesisch                    |
| na            | Nauruisch        | ne            | Nepalisch                   | nl            | Holländisch                   |
| no            | Norwegisch       | oc            | Okzitanisch                 | om            | Oromo                         |
| or            | Orija            | pa            | Pundjabisch                 | pl            | Polnisch                      |
| ps            | Paschtu          | pt            | Portugiesisch               | qu            | Quechua                       |
| rm            | Rätoromanisch    | rn            | Kirundisch                  | ro            | Rumänisch                     |
| ru            | Russisch         | rw            | Kijarwanda                  | sa            | Sanskrit                      |
| sd            | Zinti            | sg            | Sango                       | sh            | Serbokroatisch (veraltet)     |
| si            | Singhalesisch    | sk            | Slowakisch                  | sl            | Slowenisch                    |
| sm            | Samoaanisch      | sn            | Schonisch                   | so            | Somalisch                     |
| sq            | Albanisch        | sr            | Serbisch                    | ss            | Swasiländisch                 |
| st            | Sesothisch       | su            | Sudanesisch                 | sv            | Schwedisch                    |
| sw            | Suaheli          | ta            | Tamilisch                   | te            | Tegulu                        |
| tg            | Tadschikisch     | th            | Thai                        | ti            | Tigrinja                      |
| tk            | Turkmenisch      | tl            | Tagalog                     | tn            | Sezuan                        |
| to            | Tongaisch        | tr            | Türkisch                    | ts            | Tsongaisch                    |
| tt            | Tatarisch        | tw            | Twi                         | ug            | Uigur                         |
| uk            | Ukrainisch       | ur            | Urdu                        | uz            | Usbekisch                     |
| vi            | Vietnamesisch    | vo            | Volapük                     | wo            | Wolof                         |
| xh            | Xhosa            | yi            | Jiddish                     | yo            | Joruba                        |
| za            | Zhuang           | zh            | Chinesisch                  | zu            | Zulu                          |

### Übersicht zu Länderkürzeln nach ISO 3166

Falls Sie solche Länderkürzel für zusammengesetzte Sprachenkürzel benutzen, um Ländervarianten von Sprachen zu bezeichnen, also etwa en-US oder de-CH, dann notieren Sie das Länderkürzel in Großbuchstaben. In anderen Zusammenhängen, etwa bei Toplevel-Domains, werden die Länderkürzel dagegen klein geschrieben.

Die Tabelle ist nach Kürzeln alphabetisch sortiert und erhebt keinen Anspruch auf aktuelle Vollständigkeit. Das Sekretariat dieser ISO-Norm hat das Deutsche Institut für Normung e.V.



(DIN) übernommen. Dort finden Sie auch eine aktuelle Liste mit *English country names and code elements* ([www.din.de/gremien/nas/nabd/iso3166ma/codlstp1/en\\_listp1.html](http://www.din.de/gremien/nas/nabd/iso3166ma/codlstp1/en_listp1.html)).

| Kürzel | Bedeutung               | Kürzel | Bedeutung                    | Kürzel | Bedeutung                                |
|--------|-------------------------|--------|------------------------------|--------|------------------------------------------|
| AD     | Andorra                 | AE     | Vereinigte Arabische Emirate | AF     | Afghanistan                              |
| AG     | Antigua und Barbuda     | AI     | Anguilla                     | AL     | Albanien                                 |
| AM     | Armenien                | AN     | Niederländische Antillen     | AO     | Angola                                   |
| AQ     | Antarktis               | AR     | Argentinien                  | AS     | Samoa                                    |
| AT     | Österreich              | AU     | Australien                   | AW     | Aruba                                    |
| AZ     | Aserbaidschan           | BA     | Bosnien-Herzegowina          | BB     | Barbados                                 |
| BD     | Bangladesh              | BE     | Belgien                      | BF     | Burkina Faso                             |
| BG     | Bulgarien               | BH     | Bahrain                      | BI     | Burundi                                  |
| BJ     | Benin                   | BM     | Bermudas                     | BN     | Brunei                                   |
| BO     | Bolivien                | BR     | Brasilien                    | BS     | Bahamas                                  |
| BT     | Bhutan                  | BV     | Bouvet Inseln                | BW     | Botswana                                 |
| BY     | Weißrußland             | BZ     | Belize                       | CA     | Kanada                                   |
| CC     | Kokosinseln             | CF     | Zentralafrikanische Republik | CG     | Kongo                                    |
| CH     | Schweiz                 | CI     | Elfenbeinküste               | CK     | Cook Inseln                              |
| CL     | Chile                   | CM     | Kamerun                      | CN     | China                                    |
| CO     | Kolumbien               | CR     | Costa Rica                   | CS     | Tschechoslowakei (obsolet)               |
| CU     | Kuba                    | CV     | Kap Verde                    | CX     | Christmas Island                         |
| CY     | Zypern                  | CZ     | Tschechische Republik        | DE     | Deutschland                              |
| DJ     | Djibuti                 | DK     | Dänemark                     | DM     | Dominika                                 |
| DO     | Dominikanische Republik | DZ     | Algerien                     | EC     | Ecuador                                  |
| EE     | Estland                 | EG     | Ägypten                      | EH     | Westsahara                               |
| ER     | Eritrea                 | ES     | Spanien                      | ET     | Äthiopien                                |
| FI     | Finnland                | FJ     | Fidschi Inseln               | FK     | Falkland Inseln                          |
| FM     | Mikronesien             | FO     | Färöer Inseln                | FR     | Frankreich                               |
| FX     | Frankreich (nur Europa) | GA     | Gabun                        | GB     | Großbritannien (UK)                      |
| GD     | Grenada                 | GE     | Georgien                     | GF     | französisch Guyana                       |
| GH     | Ghana                   | GI     | Gibraltar                    | GL     | Grönland                                 |
| GM     | Gambia                  | GN     | Guinea                       | GP     | Guadeloupe                               |
| GQ     | Äquatorial Guinea       | GR     | Griechenland                 | GS     | South Georgia und South Sandwich Islands |
| GT     | Guatemala               | GU     | Guam                         | GW     | Guinea Bissau                            |
| GY     | Guyana                  | HK     | Hong Kong                    | HM     | Heard und McDonald Islands               |

| <i>Kürzel</i> | <i>Bedeutung</i>                   | <i>Kürzel</i> | <i>Bedeutung</i>           | <i>Kürzel</i> | <i>Bedeutung</i>                  |
|---------------|------------------------------------|---------------|----------------------------|---------------|-----------------------------------|
| HN            | Honduras                           | HR            | Kroatien                   | HT            | Haiti                             |
| HU            | Ungarn                             | ID            | Indonesien                 | IE            | Irland                            |
| IL            | Israel                             | IN            | Indien                     | IO            | Britisch-Indischer<br>Ozean       |
| IQ            | Irak                               | IR            | Iran                       | IS            | Island                            |
| IT            | Italien                            | JM            | Jamaika                    | JO            | Jordanien                         |
| JP            | Japan                              | KE            | Kenia                      | KG            | Kirgisistan                       |
| KH            | Kambodscha                         | KI            | Kiribati                   | KM            | Komoren                           |
| KN            | St. Kitts Nevis<br>Anguilla        | KP            | Nord Korea                 | KR            | Süd Korea                         |
| KW            | Kuwait                             | KY            | Kaiman Inseln              | KZ            | Kasachstan                        |
| LA            | Laos                               | LB            | Libanon                    | LC            | Saint Lucia                       |
| LI            | Liechtenstein                      | LK            | Sri Lanka                  | LR            | Liberia                           |
| LS            | Lesotho                            | LT            | Litauen                    | LU            | Luxemburg                         |
| LV            | Lettland                           | LY            | Libyen                     | MA            | Marokko                           |
| MC            | Monaco                             | MD            | Moldavien                  | MG            | Madagaskar                        |
| MH            | Marshall Inseln                    | MK            | Mazedonien                 | ML            | Mali                              |
| MM            | Birma                              | MN            | Mongolei                   | MO            | Macao                             |
| MP            | Marianen                           | MQ            | Martinique                 | MR            | Mauretanien                       |
| MS            | Montserrat                         | MT            | Malta                      | MU            | Mauritius                         |
| MV            | Malediven                          | MW            | Malawi                     | MX            | Mexiko                            |
| MY            | Malaysia                           | MZ            | Mocambique                 | NA            | Namibia                           |
| NC            | Neukaledonien                      | NE            | Niger                      | NF            | Norfolk Inseln                    |
| NG            | Nigeria                            | NI            | Nicaragua                  | NL            | Niederlande                       |
| NO            | Norwegen                           | NP            | Nepal                      | NR            | Nauru                             |
| NU            | Niue                               | NZ            | Neuseeland                 | OM            | Oman                              |
| PA            | Panama                             | PE            | Peru                       | PF            | französisch<br>Polynesien         |
| PG            | Papua Neuguinea                    | PH            | Philippinen                | PK            | Pakistan                          |
| PL            | Polen                              | PM            | St. Pierre und<br>Miquelon | PN            | Pitcairn                          |
| PR            | Puerto Rico                        | PT            | Portugal                   | PW            | Palau                             |
| PY            | Paraguay                           | QA            | Qatar                      | RE            | Reunion                           |
| RO            | Rumänien                           | RU            | Russland                   | RW            | Ruanda                            |
| SA            | Saudi Arabien                      | SB            | Solomon Inseln             | SC            | Seychellen                        |
| SD            | Sudan                              | SE            | Schweden                   | SG            | Singapur                          |
| SH            | St. Helena                         | SI            | Slowenien                  | SJ            | Svalbard und Jan<br>Mayen Islands |
| SK            | Slowakei (slowakische<br>Republik) | SL            | Sierra Leone               | SM            | San Marino                        |
| SN            | Senegal                            | SO            | Somalia                    | SR            | Surinam                           |
| ST            | Sao Tome                           | SU            | Sowjet- Union<br>(obsolet) | SV            | El Salvador                       |

| <i>Kürzel</i> | <i>Bedeutung</i>             | <i>Kürzel</i> | <i>Bedeutung</i>               | <i>Kürzel</i> | <i>Bedeutung</i>        |
|---------------|------------------------------|---------------|--------------------------------|---------------|-------------------------|
| SY            | Syrien                       | SZ            | Swasiland                      | TC            | Turks und Kaikos Inseln |
| TD            | Tschad                       | TF            | Französisches Süd Territorium  | TG            | Togo                    |
| TH            | Thailand                     | TJ            | Tadschikistan                  | TK            | Tokelau                 |
| TM            | Turkmenistan                 | TN            | Tunesien                       | TO            | Tonga                   |
| TP            | Timor                        | TR            | Türkei                         | TT            | Trinidad Tobago         |
| TV            | Tuvalu                       | TW            | Taiwan                         | TZ            | Tansania                |
| UA            | Ukraine                      | UG            | Uganda                         | UK            | Großbritannien          |
| UM            | US kleinere Inseln außerhalb | US            | Vereinigte Staaten von Amerika | UY            | Uruguay                 |
| UZ            | Usbekistan                   | VA            | Vatikan                        | VC            | St. Vincent             |
| VE            | Venezuela                    | VG            | Virgin Island (Brit.)          | VI            | Virgin Island (USA)     |
| VN            | Vietnam                      | VU            | Vanuatu                        | WF            | Wallis et Futuna        |
| WS            | Samoa                        | YE            | Jemen                          | YT            | Mayotte                 |
| YU            | Jugoslawien (obsolete)       | ZA            | Südafrika                      | ZM            | Sambia                  |
| ZR            | Zaire                        | ZW            | Zimbabwe                       |               |                         |

### **Beispiele für zusammengesetzte Sprachenkürzel**

Die folgenden Beispiele zeigen ein paar Beispiele für zusammengesetzte Sprachenkürzel, wie sie nach RFC 1766 erlaubt sind:

| <i>Kürzel</i> | <i>Bedeutung</i>              |
|---------------|-------------------------------|
| de-CH         | Schweizerdeutsch              |
| de-AT         | österreichisches Deutsch      |
| en-US         | amerikanisches Englisch       |
| en-CA         | kanadisches Englisch          |
| en-AU         | australisches Englisch        |
| en-GB         | britisches Englisch           |
| en-UK         | königlich-britisches Englisch |
| cel-gaulish   | Gallisch                      |
| fr-CA         | kanadisches Französisch       |
| zh-TW         | Taiwan-Chinesisch             |
| zh-guoyu      | Mandarin-Chinesisch           |
| zh-min-nan    | Min-Nan-Chinesisch            |
| zh-xiang      | Xiang-Chinesisch              |

## A.4 Client-Identifikationen

### *Allgemeines zu Client-Identifikationen*

Wenn ein Web-Browser oder ein Suchrobot oder der Agent eines Proxy-Servers bei einem Web-Server Daten anfordert, schickt er immer auch eine Zeichenkette mit, mit der er sich dem Web-Server gegenüber »namentlich vorstellt«. Ein einheitliches oder standardisiertes Schema dafür ist in der Spezifikation zu HTTP 1.1 definiert worden. Nach der Ankündigung der Vorstellung eines »User-Agents« (das ist der Oberbegriff für die anfangs genannten Programme, die Daten anfordern können) wird wenigstens ein Hinweis auf das Produkt oder ein Kommentar übersandt. Nicht definiert ist die für die Praxis notwendige Unterscheidung zwischen verschiedenen Produktversionen oder -typen. Das ist bedauerlich. In der Praxis hat sich eine gewisse Regel herausgeschält: Produktname und Produktversion werden in dieser Reihenfolge und durch Schrägstrich getrennt dargestellt. Es halten sich allerdings nicht alle Browser an dieses Schema. Einige Browser-Typen geben sich als andere aus, und wieder andere erlauben es auch dem Anwender an der Programmoberfläche, eine beliebige Identifikation einzustellen. Wenn der Anwender dann seinen Namen oder irgendeinen Unsinn eingibt, meldet sich der Browser damit beim Server an.

Dennoch wird die Client-Identifikation an verschiedenen Stellen gespeichert und kann in manchen Situationen zu nützlichen Zwecken dienen:

- In JavaScript gibt es das `navigator`-Objekt, das es beispielsweise erlaubt, so genannte »Browser-Weichen« zu programmieren, um bestimmte JavaScript-Anweisungen nur auszuführen, wenn bestimmte Browser-Typen am Werk sind.
- Ein CGI-Script kann die CGI-Umgebungsvariable `HTTP_USER_AGENT` auslesen, um herauszufinden, von welcher Client-Software es aufgerufen wurde. Auch dabei kann die Information dazu genutzt werden, um bestimmte Anweisungen abhängig auszuführen oder nicht auszuführen.
- Bei Statistiken zu einem Web-Projekt gibt es in der Regel auch eine Statistiksorte, die Zugriffszahlen nach Browser-Typen sortiert aufschlüsselt. Grundlage dafür sind ebenfalls die Client-Identifikationen, die der Web-Server bei den Aufrufen erhalten hat.

### *Client-Identifikationen von Netscape-Browsern*

Netscape gibt sich seit jeher mit dem Namen `Mozilla` aus. Eindeutig erkennbar ist er daran allerdings überhaupt nicht, denn der Internet Explorer gibt sich beispielsweise ebenfalls als `Mozilla` aus. Die folgenden Beispiele zeigen einige typische Identifikationen, die jedoch nur für Netscape typisch sind.

- `Mozilla/2.0(Macintosh;I;PPC)`
- `Mozilla/2.0(Win16;I)`
- `Mozilla/3.01Gold[de](Win16;I)`
- `Mozilla/4.03 [en] (X11; I; Linux 2.0.30 i686)`
- `Mozilla/4.01 [en] (Win95; I)`
- `Mozilla/4.01 [en]C-CCK-MCD (WinNT; I)`

- Mozilla/4.01 [en]C-FedExIntl (Win95; I)
- Mozilla/4.5 [en] (X11; I; SunOS 5.6 sun4d)
- Mozilla/4.6 [en] (X11; I; FreeBSD 3.2-RELEASE i386)
- Mozilla/5.0 [en] (WinNT; I)
- Mozilla/5.0b1 [en] (X11; N; Linux 2.2.0 alpha; Nav)

Das grundsätzliche Schema für Produktname und Version wird also eingehalten – die Identifikation startet immer mit `Mozilla/[Versionsangabe]`. Damit ist er allerdings noch nicht von vielen anderen Browsern unterscheidbar! Die Versionsangabe kann auch Zusatzangaben enthalten wie `5.0b1` (was eigentlich für Version 6.0 steht, da Version 5.0 übersprungen wurde, und `b1` steht für »Betaversion 1«). Auch andere Zusätze wie `3.01Gold` können vorkommen (`Gold` stand seinerzeit für die erste Variante des Netscape-Browsers, die zugleich Wysiwyg-Editieren erlaubte, also ein Vorläufer des Netscape Composer).

Hinter der Versionsangabe kann in eckigen Klammern ein Sprachenkürzel folgen, das die Sprache der Browser-Bedienelemente angibt. Das Sprachenkürzel ist bei neueren Versionen zwar Standard, aber es kann auch mal fehlen. In runden Klammern folgen anschließend weitere Informationen. Mehrere davon werden durch Semikolon voneinander getrennt. Zunächst wird die verwendete Plattform genannt. Dabei sind Abkürzungen wie `Win95`, `Win98` oder `WinNT` für entsprechende Windows-Versionen möglich. Bei allen Unix-basierten Systemen wird an der Stelle meistens `X11` angegeben, was für X-Windows steht, und bei Mac-Rechnern `Macintosh`.

An zweiter Stelle innerhalb der runden Klammern folgt ein Großbuchstabe, der das Sicherheits-Level des Browsers bezeichnet (Länge von Schlüsseln). Ein `U` steht für »US-konform«, ein `I` für »international«. Wofür das `N` im letzten der obigen Beispiele steht, war nicht zu ermitteln. Hinter dem Sicherheitskennzeichen folgen dann noch beschreibende Informationen zum verwendeten Prozessortyp und/oder nähere Angaben zum Betriebssystem. Alle Teile können Leerzeichen dazwischen haben, aber zwingend ist dies nicht.

### ***Client-Identifikationen des MS Internet Explorers***

Der Internet Explorer gibt sich in seiner Browser-Identifikation deutlich weniger selbstbewusst als sonst. Er versteckt sich stets hinter dem gleichen Codenamen wie die Netscape-Browser, gibt zunächst eine Versionsnummer an, die sich auf Netscape bezieht, und rückt dann, durch das Schlüsselwort `compatible` eingeleitet, mit den Informationen über sich selber heraus. Die folgenden Beispiele zeigen einige typische Identifikationen des Internet Explorers.

- Mozilla/1.22(compatible;MSIE2.01;WindowsNT)
- Mozilla/2.0(compatible;MSIE3.0;AK;Windows95)
- Mozilla/2.0(compatible;MSIE3.02;Updatea;AK;AOL3.0;Windows95)
- Mozilla/4.0 (compatible; MSIE 4.0; Windows 95;BPH01)
- Mozilla/4.0 (compatible; MSIE 4.0; Mac\_68000)
- Mozilla/4.0 (compatible; MSIE 4.0; Mac\_PowerPC; e412354MCICDV4.0)
- Mozilla/4.0 (compatible; MSIE 4.0b2; Windows 95)

- Mozilla/4.0 (compatible; MSIE 5.0; AOL 4.0; Windows 95)
- Mozilla/4.0 (compatible; MSIE 5.0; MSN 2.5; Windows 98)
- Mozilla/4.0 (compatible; MSIE 5.0; Windows 95)

Das grundsätzliche Schema für Produktname und Version folgt dem von Netscape und wird also eingehalten – die Identifikation startet immer mit `Mozilla/[Versionsangabe]`. Die Versionsangabe dahinter bedeutet, zu welcher Netscape-Browser-Version sich der ausgewiesene Internet Explorer für voll kompatibel hält. Das ist natürlich immer mit gewissen Vorbehalten zu genießen, denn ein Internet Explorer hat beispielsweise zu keinem Zeitpunkt je das `layer`-Element in HTML interpretiert, also ist die Behauptung der Kompatibilität zu Netscape 4.0 nicht unbedingt haltbar. Von daher bringt die Versionsangabe hinter dem `Mozilla/` beim Internet Explorer keine wirkliche Erkenntnis.

Seine eigentliche Identität verrät der Internet Explorer erst in den runden Klammern. Diese enthalten normalerweise mehrere Angaben, durch Semikolon getrennt. Der erste Teil ist stets das Schlüsselwort `compatible`. Dahinter folgen die vier Großbuchstaben `MSIE`, die eigentlich den Bezug zum Internet Explorer herstellen. Dahinter folgt die genaue Versionsangabe.

Alle weiteren Angaben können differieren und eine unterschiedliche Reihenfolge haben. Bei Betriebssystembezeichnungen fällt auf, dass `Windows` durchweg ausgeschrieben ist, während `Macintosh`-Versionen des Internet Explorers sich üblicherweise mit `Mac` plus Unterstrich plus nähere Bezeichnung ausgeben. Wenn es sich um eine Browser-Distribution eines Online-Dienstes wie AOL oder MSN handelt, wird dies in einem Zusatzeintrag wie `AOL 4.0` verzeichnet. Alle Teile können Leerzeichen dazwischen haben, aber zwingend ist dies nicht, besonders bei älteren Versionen nicht.

## A.5 Client-Identifikationen anderer Browser

Die folgenden Beispiele zeigen typische Identifikationen einiger anderer Browser-Typen.

- Mozilla/1.2 (Opera/2.02)
- Mozilla/3.0 (compatible; Opera/3.0; Windows 95) 3.51
- Mozilla/3.0 (Windows 4.10;US) Opera 3.60 [en]
- DosLynx/0.8a
- Lynx 2.4-FM
- Lynx 2.5 libwww-FM/2.14
- Lynx 2.9 (compatible Interactive Tcl/Tk News Ticker v0.06)
- Lynx/2-4-2 MU-MODS
- Lynx\_ALynx/2.4 AmigaPort/1.29
- amaya/V2.0a libwww/5.2.1
- AIR\_Mosaic(16bit)/v3.10.198.18

- IBM WebExplorer DLL /v1.1
- IBrowse/1.2 (AmigaOS 3.1)
- iCab/Pre1.4 (Macintosh; I; PPC)

Die Beispiele zeigen, wie uneinheitlich die unterschiedlichen Browser-Typen sich identifizieren, was die Auswertung nicht gerade erleichtert. In der Regel hilft beim Analysieren beispielsweise mit einem Script nur, im ersten Schritt nach einschlägigen Schlüsselwörtern wie *Opera*, *Mosaic*, *amaya*, *Lynx* oder *ICab* zu suchen. Im zweiten Schritt können Sie dann versuchen, durch Suchen nach Teilausdrücken wie *Win*, *Mac*, *Amiga* usw. das Betriebssystem herauszufinden.

### ***Client-Identifikationen von Such-Robots und Proxy-Agenten***

Es gibt kein Patentrezept, um »menschliche Seitenbesucher« von Softwareprogrammen zu unterscheiden, die beim Aufrufen der Web-Seiten im Batchbetrieb arbeiten und beispielsweise die Daten für ihre Suchmaschine indizieren oder Dateien an einen Proxy-Rechner übertragen. Die folgenden Beispiele zeigen ein paar typische Identifikationen solcher Programme.

- Eule Robot v3.00 (www.eule.de)
- FAST-WebCrawler/0.3 (crawler@fast.no)
- Googlebot/1.0 (googlebot@googlebot.com http://googlebot.com/)
- JCrawler/0.3 (robot; webmaster@netimages.com)
- Netscape-Proxy/3.52 (Batch update)
- Openfind Robot/1.1A2
- W3CRobot/4.0D
- WebCrawler/2.0
- Lycos\_Spider\_(T-Rex)/3.0
- Aladin/3.324
- taz.northernlight.com Gulliver/1.3
- Openfind Robot/1.1A2
- KIT-Fireball/2.0
- URL-Minder/2.4.5

Bei Suchmaschinen-Robots gibt es zumindest einige Anhaltspunkte, nach denen sich eine Identifikationszeichenkette durchsuchen lässt. Solche Robots identifizieren sich oft mit Schlüsselwörtern wie *Robot*, *Spider* oder *Crawler*. Viele von ihnen tragen im Namen auch den Namen des zugehörigen Suchdienstes. In den obigen Beispielen sind etwa *Google*, *Fireball*, *Lycos*, *Eule*, *Northernlight* und *Aladin* erkennbar. Generell sind Robots und auch Agenten von Proxy-Servern häufig daran erkennbar, dass sie bei der Identifikation einen Domain-Namen, einen URI oder eine E-Mail-Adresse angeben.

## A.6 robots.txt – Robots kontrollieren

### Allgemeines zur robots.txt

Es gibt De-facto-Standards im Internet, die einfach gewachsen sind, ohne es je zu einer RFC gebracht zu haben. Dazu gehört auch der Status, den die Datei *robots.txt* im Web hat. In einer Datei dieses Namens können Betreiber von Web-Projekten angeben, welcher Suchrobot welche Projektverzeichnisse auslesen darf und welcher was nicht lesen darf. Die Datei enthält also Anweisungen für Robots von Suchmaschinen. Die überwiegende Mehrheit der Robots moderner Suchmaschinen berücksichtigen das Vorhandensein einer *robots.txt*, lesen sie aus und befolgen die Anweisungen.

Zwar lässt sich auch in einzelnen HTML-Dateien mit Hilfe eines Meta-Tags für Suchprogramme das Auslesen erlauben bzw. verbieten (Kapitel 4.1.2). Doch das betrifft nur die jeweilige HTML-Datei und maximal alle weiteren, durch Verweise erreichbaren Dateien. In einer zentralen *robots.txt* können Sie dagegen unabhängig von der Datei- und Verweisstruktur Ihres Web-Projekts festlegen, welche Verzeichnisse und Verzeichnisbäume ausgelesen werden dürfen und welche nicht.

### Speicherort und Aufbau einer robots.txt

Die *robots.txt* muss unter diesem Namen (alle Buchstaben klein geschrieben) im Wurzelverzeichnis der Web-Dateien der Domain abgelegt werden. Wenn Sie also einen Domain-Namen *mein-name.de* haben, dann muss die *robots.txt* in dem Verzeichnis abgelegt werden, in dem auch die oberste Einstiegsdatei von *www.mein-name.de* liegt. Der URI wäre also *http://www.mein-name.de/robots.txt*. Nur so kann sie von Suchmaschinen-Robots, die das Projekt aufsuchen, gefunden werden. Das bedeutet, dass Sie die Technik der *robots.txt* nur nutzen können, wenn Sie eine eigene Domain haben, nicht aber bei Webspace-Angeboten, wo Sie lediglich ein Homepage-Verzeichnis auf einem Server erhalten, ohne an das Wurzelverzeichnis der Domain zu kommen. Die *robots.txt* ist eine reine Textdatei und muss mit einem Texteditor bearbeitet werden.

Beispiel: *robots.txt*:

```
# robots.txt zu http://www.mein-name.de/

User-agent: UniservalRobot/1.0
User-agent: mein-Robot
Disallow: /quellen/dtd/
Allow: /quellen/dtd/index.html

User-agent: *
Disallow: /unsinn/
Disallow: /temp/
Disallow: /newsticker.shtml
```

Die erste Zeile ist lediglich eine Kommentarzeile. Kommentarzeilen werden durch ein Gatterzeichen # eingeleitet. Ansonsten besteht eine *robots.txt* aus Datensätzen (*records*). Im obigen Beispiel sind zwei solcher Datensätze notiert. Ein Datensatz besteht grundsätzlich aus zwei Teilen. Im ersten Teil wird angegeben, für welche Robots (*User-agent*) die nachfolgenden Anweisun-



gen gelten. Im zweiten Teil werden die Anweisungen selbst notiert. Die Anweisungen bestehen darin, den zuvor bestimmten Robots etwas zu verbieten (*Disallow*) oder zu erlauben (*Allow*).

Jede Zeile eines Datensatzes beginnt mit einem der drei erlaubten Schlüsselwörter *User-agent*, *Disallow* oder *Allow*. Dahinter folgt, durch ein Doppelpunkt und Leerzeichen getrennt, die zugehörige Angabe. Zwischen den Datensätzen wird eine Leerzeile notiert. Innerhalb eines Datensatzes muss zunächst mindestens eine Zeile mit *User-agent:* beginnen. Dahinter ist immer nur eine Angabe möglich. Wenn Sie mehr als einen bestimmten Robot ansprechen möchten, müssen Sie mehrere Zeilen untereinander notieren, die mit *User-agent:* beginnen – so wie im ersten Datensatz des obigen Beispiels. Unterhalb der Zeilen, die mit *User-agent:* beginnen, werden die Zeilen notiert, die mit *Disallow:* oder *Allow:* beginnen. Die Angaben dazu werden dann von den Robots beachtet, die im gleichen Datensatz mit *User-agent* spezifiziert wurden.

Bei *User-agent:* ist entweder die Angabe *\** (Stern-Zeichen) erlaubt, was so viel bedeutet wie »alle Robots«, oder der Name eines bestimmten Robots. Diesen Namen müssen Sie allerdings kennen. Wenn Sie *\** angeben, sollte in dem gleichen Datensatz keine weitere Angabe zu *User-agent* folgen, da dies dem Platzhalter für »alle« widersprechen würde.

Hinter jeder Zeile, die mit *Disallow:* oder *Allow:* beginnt, können Sie jeweils eine Pfadangabe notieren. Benutzen Sie im Zweifelsfall so viel wie möglich die Angabe *Disallow:* und so wenig wie möglich die Angabe *Allow:*. Der Grund ist, dass *Disallow:* früher in die Spezifikation der *robots.txt* aufgenommen wurde und von mehr Robots verstanden wird als *Allow:*.

Bei den Angaben zu *Disallow:* und *Allow:* können Sie Verzeichnispfade und einzelne Dateien mit Pfadangabe angeben. Wildcards wie *\** oder *\*.\** sind dabei **nicht** erlaubt. Achten Sie darauf, bei Verzeichnispfaden einen abschließenden Schrägstrich */* zu notieren. Wenn Sie nämlich beispielsweise */index* notieren, wäre auch die Datei */index.html* betroffen, und nicht nur das Unterverzeichnis */index/*.

Im ersten der obigen Beispieldatensätze wird angenommen, dass ein superschlauer Robot namens *UniversalRobot/1.0* sowie ein selbst gestrickter Robot namens *mein-Robot* Daten aus dem Verzeichnis */quellen/dtd/* in ihre Suchmaschinen einspeisen. Das ist aber nicht erwünscht, weil in diesem Verzeichnis beispielsweise DTDs für XML-Dateien abgespeichert sind. Deshalb wird speziell diesen beiden Robots der Zugriff auf dieses Verzeichnis (und alle Unterverzeichnisse davon) verboten. Nur die Datei *index.html*, die im Verzeichnis für die DTDs ebenfalls abgelegt ist, dürfen die beiden Robots trotzdem auslesen.

Im zweiten Datensatz wird allen Robots verboten, die beiden Unterverzeichnisse */unsinn/* und */temp/* auszulesen. Die Verzeichnisnamen sprechen ja für sich, und es ist leicht ersichtlich, wozu dieses Verbot gut ist: nämlich um überflüssigen Datenmüll in den großen Suchmaschinen zu vermeiden. Ferner wird der Zugriff auf die Datei *newsticker.shtml* verboten. Der Grund könnte sein, dass diese Datei von einem über Server Side Includes eingebundenen CGI-Script laufend aktuelle Daten erhält, weswegen es keinen Sinn macht, diese Datei in Suchmaschinen aufzunehmen.

**Beachten Sie:** Mit der folgenden Syntax erlauben Sie keinem Suchmaschinen-Robot, auch nur irgendetwas von Ihren Daten auszulesen. Mit */* bestimmen Sie »alle Daten dieses Verzeichnisses und aller Unterverzeichnisse«:

```
User-agent: *
Disallow: /
```

Mit der folgenden Syntax nehmen Sie einen bestimmten Robot namens `mein-Robot` von allen anderen Verboten aus:

```
User-agent: mein-Robot
Disallow:
```

Durch eine fehlende Angabe hinter `Disallow:` wird alles erlaubt! Web-Browser ignorieren die *robots.txt*. Es ist also nicht möglich, damit Daten vor Anwendern zu schützen. Lesen Sie zu diesem Zweck den folgenden Abschnitt *.htaccess – Server-Reaktionen kontrollieren*.

## A.7 *.htaccess – Server-Reaktionen kontrollieren*

### **Allgemeines zu *.htaccess*-Dateien**

*.htaccess*-Dateien sind Konfigurationsdateien für Verzeichnisse, die zum Web-Angebot gehören. So ist die *.htaccess*-Technik beispielsweise der übliche Weg, um nur bestimmten Benutzern den Zugriff auf bestimmte Daten zu erlauben. *.htaccess* ist also die Antwort auf die immer wieder gestellte Frage nach einem »richtigen« Passwortschutz. Doch mit *.htaccess*-Dateien können Sie noch viel mehr anstellen, als Daten mit Passwörtern zu schützen. So können Sie auch ganze Benutzerkreise automatisch (ohne Passwortzugang) aussperren oder alle bis auf bestimmte aussperren. Sie können Optionen zum so genannten Verzeichnis-Browsing (Web-Browser ruft ein Verzeichnis auf, in dem keine Default-Datei wie z.B. *index.htm* liegt) einstellen. Sie können automatische Weiterleitungen einstellen oder eigene Regelungen für den Fall von HTTP-Fehlermeldungen schaffen. Sie können sogar abhängig von bestimmten Bedingungen alternative Inhalte servieren, beispielsweise Web-Seiten mit unterschiedlichen Landessprachen abhängig von der Sprache des benutzten Web-Browsers – ohne CGI, ohne JavaScript. Und schließlich können Sie sogar einstellen, dass Daten komprimiert an den aufrufenden Browser übertragen werden.

Web-Server, die zum NCSA-Server kompatibel sind, kennen das Konzept der *.htaccess*-Dateien. Heutzutage trifft das vor allem auf den immer dominanter werdenden und die De-facto-Standards im Web setzenden Web-Server **Apache** zu. Wenn auf Ihrem Server-Rechner also ein Apache Web-Server läuft, dann können Sie die Technik der *.htaccess*-Dateien einsetzen. Falls Sie nicht wissen, welche Web-Server-Software auf dem Rechner läuft, wo Sie Ihr Web-Projekt hosten, dann können Sie im Online-Angebot von SELFHTML bei [selfhtml.teamone.de](http://selfhtml.teamone.de) aktuell den SELFHTML Server Watch benutzen, um es herauszufinden. Fragen Sie im Zweifelsfall Ihren Provider oder Webmaster!

Dateien mit dem Namen *.htaccess* (das erste Zeichen ist ein Punkt) sind im Grunde Teil der Konfiguration des Web-Servers. Der Web-Server verfügt zwar auch über zentrale Konfigurationsdateien. Doch diese liegen normalerweise unterhalb des Programmverzeichnis des Web-Servers und sind nicht unbedingt den Benutzern zugänglich, die ein Web-Projekt verwalten. *.htaccess*-Dateien werden dagegen in dem Verzeichnisbereich abgelegt, in dem das Web-Angebot liegt. Also dort, wo Sie Ihre HTML-Dateien, Grafikdateien usw. haben, können Sie also auch *.htaccess*-Dateien ablegen.

Dass der Name *.htaccess* mit einem Punkt beginnt, ist eine Tradition aus der Unix-Welt. Dort beginnen viele verzeichnisspezifische Konfigurationsdateien mit einem Punkt. Bekannt ist beispielsweise die Datei *.profile*. In der zentralen Konfiguration des Apache Web-Servers lässt sich

allerdings auch ein anderer Dateiname als *.htaccess* einstellen. *.htaccess*-Dateien sind reine Textdateien, die mit einem Texteditor bearbeitet werden müssen. Wenn Sie Telnet- oder SSH-Zugang zu dem Server-Rechner haben, auf dem Ihr Web-Angebot liegt, können Sie solche Dateien direkt auf dem Server erstellen und bearbeiten, indem Sie dort einen geeigneten Texteditor aufrufen (auf Unix-Systemen z.B. den *vi*-Editor). Wenn Sie nur FTP-Zugang zu dem Server-Rechner haben, können Sie eine solche Datei auf Ihrem lokalen PC mit einem Texteditor erstellen und dann hochladen.

Sie können alle in diesem Abschnitt einzeln vorgestellten Möglichkeiten auch in einer *.htaccess*-Datei kombinieren. So können Sie also beispielsweise zugleich einen Passwortschutz einrichten und gleichzeitig Optionen zum Verzeichnis-Browsing oder für das Senden alternativer Inhalte notieren.

*.htaccess*-Dateien gelten stets für das Verzeichnis, in dem Sie abgespeichert werden, sowie für alle Unterverzeichnisse davon und für deren Unterverzeichnisse. Wenn Sie in einem der Unterverzeichnisse andere Regelungen wünschen, müssen Sie dort wiederum eine *.htaccess*-Datei ablegen. Die Angaben der jeweils aktuellen Datei überschreiben die Angaben aus den Verzeichnissen oberhalb.

### **Verzeichnisse und Dateien mit Passwort schützen**

*.htaccess*-Dateien sind verzeichnispezifisch. Speichern Sie die *.htaccess*-Datei also in dem Verzeichnis ab, in dem die geschützten Daten liegen. Sie können wahlweise das ganze Verzeichnis mit all seinen Unterverzeichnissen oder nur bestimmte Dateien oder Dateitypen schützen. Sie können den Passwortschutz außerdem wahlweise für einzelne Benutzer oder für ganze Benutzergruppen einrichten. Auch Kombinationen beider Formen sind möglich. Damit der Verzeichnisschutz mit Passwort funktioniert, genügt die *.htaccess*-Datei alleine allerdings nicht. Sie brauchen zusätzlich eine Datei, in der die Benutzer und ihre Passwörter stehen. Falls Sie mit Benutzergruppen arbeiten, benötigen Sie außerdem noch eine Datei, in der die Benutzergruppen definiert werden. Auch diese beiden anderen Dateien werden mit einem Texteditor erstellt.

Beispiel: *.htaccess*-Datei

```
# .htaccess-Datei fuer Web-Verzeichnis /service
AuthType Basic
AuthName "Service-Bereich"
AuthUserFile /usr/verwaltung/web/.htusers
AuthGroupFile /usr/verwaltung/web/.htgroups
require user Werner Dieter Heidi
require group Servicetechniker
```

Die erste Zeile der Beispieldatei ist lediglich ein Kommentar. Solche Kommentarzeilen müssen mit dem Gatterzeichen *#* eingeleitet werden. Alle anderen Zeilen haben einen einheitlichen Aufbau: Sie beginnen mit Schlüsselwörtern, und dahinter folgen, durch ein oder mehrere Leerzeichen getrennt, Angaben dazu. Leerzeilen sind erlaubt. Zeilen mit Angaben, die der Web-Server nicht interpretieren kann, führen jedoch möglicherweise zu einem »Internal Server Error« (HTTP-Statuscode 500).

Um einen Passwortschutz einzurichten, brauchen Sie die Schlüsselwörter *AuthType*, *AuthName*, *AuthUserFile*, und wenn Sie mit Benutzergruppen arbeiten, auch *AuthGroupFile*. Ferner

benötigen Sie eine oder mehrere Angaben zu dem Schlüsselwort `require`. Mit `AuthType` wird die Art der Authentifizierung bezeichnet. Die übliche Angabe ist `Basic` – dabei stehen die Benutzer und Passwörter in einer noch anzugebenden Datei. Ein anderer denkbarer Authentifizierungstyp wäre `PGP` (Pretty good privacy).

Bei `AuthUserFile` wird die Datei angegeben, in der die autorisierten Benutzer und ihre Passwörter stehen. Es muss der vollständige absolute Pfadname angegeben werden, also **nicht** der Pfad ab dem Wurzelverzeichnis Ihres Web-Projekts, sondern ab dem Wurzelverzeichnis des Server-Rechners. Fragen Sie dazu gegebenenfalls Ihren Provider oder Webmaster! Im Interesse der Sicherheit ist es auch besser, die Datei mit den Benutzern und Passwörtern außerhalb des Web-Projekts auf dem Server abzulegen. Aber nicht bei allen Hosting-Angeboten können Sie auf den Server-Rechner außerhalb des eigenen Web-Projekts zugreifen. In diesem Fall sollten Sie für die Datei auf jeden Fall einen Dateinamen wählen, der mit `.ht` beginnt. Denn im Apache-Server werden solche Dateien per Voreinstellung vor Besuchern verborgen. Das Gleiche gilt für die Gruppdatei, falls Sie mit Benutzergruppen arbeiten. Diese wird mit `AuthGroupFile` angegeben.

Bei `require` geben Sie als zweites Schlüsselwort entweder `user` oder `group` an, je nachdem, ob Sie einzelne Benutzer oder Benutzergruppen meinen. Dahinter können ein oder mehrere Namen von Benutzern oder Benutzergruppen folgen. Im obigen Beispiel werden die drei Benutzer `Werner`, `Dieter`, `Heidi` sowie alle Benutzer der Gruppe `Servicetechniker` angegeben. Damit der Passwortschutz funktioniert, müssen nun die angegebenen Dateien mit den Benutzern und den Gruppen angelegt werden.

Beispiel: *.htusers*-Datei unter Unix

```
# Benutzerdatei fuer Web-Projekt
Werner:INY8//m5KMwIc
Manfred:Dionysos
Janine:INw2mPEH.owe2
Anke:INh6DHvyejvf2
Bernd:INboWuvjjwQ7E
Karin:INwOX0z96UQ0U
Christina:INXo9kh0M.anc
Andreas:INeRD/cUQIFP6
Dieter:INUnlKdkNZ6RQ
Heidi:IN20ffIEEV1H6
```

Beispiel: *.htusers*-Datei z.B. unter Windows

```
# Benutzerdatei fuer Web-Projekt
Werner:Herakles
Manfred:Dionysos
Janine:Olympia
Anke:Hydra
Bernd:Poseidon
Karin:Hera
Christina:Athena
Andreas:Apollon
Dieter:Archilles
Heidi:Diana
```

Auch in solchen Dateien sind Kommentarzeilen erlaubt, eingeleitet durch `#`. Ansonsten enthält jede Zeile der Benutzerdatei einen Benutzernamen, und gleich dahinter, durch einen Doppelpunkt getrennt, das Passwort. Bei Unix-Systemen sind das aber nicht die Passwörter selbst, sondern die verschlüsselten Passwörter. Dazu müssen Sie die Passwörter mit Hilfe des `crypt`-Mechanismus verschlüsseln.

Beispiel: *.htgroups*-Datei

```
# Gruppendatei fuer Web-Projekt
Servicetechniker: Andreas Karin Janine
```

Gruppendateien bestehen aus Einträgen, bei denen zunächst ein Gruppenname notiert wird und dahinter, nach einem Doppelpunkt, die Namen von Benutzern, die zu dieser Gruppe gehören. Es müssen Benutzernamen sein, für die in der Benutzerdatei ein Eintrag angelegt wurde.

Die Gruppendatei wird im Beispiel nur benötigt, weil in der *.htaccess*-Datei eine Benutzergruppe angegeben wurde. Während Sie die Benutzerdatei in jedem Fall brauchen, ist die Gruppendatei nur erforderlich, wenn Sie Gruppennamen benutzen.

**Effekt:** Alle Besucher des Web-Projekts, die nun versuchen, auf das Verzeichnis mit der *.htaccess*-Datei zuzugreifen, bekommen von ihrem Browser einen Dialog angeboten, in dem sie Benutzernamen und Passwort eingeben müssen. Nur Besucher, die sich mit einer gültigen Kombination aus Benutzernamen und Passwort anmelden, haben Zugriff auf das Verzeichnis.

**Erweiterte Möglichkeiten:** So wie im obigen Beispiel gezeigt, gilt der Zugangsschutz für das Verzeichnis, in dem die *.htaccess*-Datei liegt, und für alle Verzeichnisse unterhalb davon. Sie können den Schutz aber auch auf bestimmte Dateien, Dateitypen oder Zugriffsmethoden einschränken.

Beispiel: *.htaccess*-Datei

```
# .htaccess-Datei fuer Web-Verzeichnis /service
AuthType Basic
AuthName "Service-Bereich"
AuthUserFile /usr/verwaltung/web/.htusers
AuthGroupFile /usr/verwaltung/web/.htgroups
<Files *.htm>
require user Werner Dieter Heidi
require group Servicetechniker
</Files>
```

Um den Schutz einzuschränken, benutzen Sie ähnlich wie in HTML oder XML Tags mit spitzen Klammern. Im einleitenden Tag kann hinter der öffnenden spitzen Klammer entweder `Files` stehen, wie im obigen Beispiel, oder `Limit`. Dahinter können Sie genau eine einschränkende Angabe machen. Mit `*.htm` wie im Beispiel beschränken Sie den Schutz auf HTML-Dateien. Mit einer Angabe wie `geheim.htm` würde nur diese eine Datei geschützt. Bei `Limit` können Sie die Zugriffsmethode z.B. auf `GET` oder `POST` einschränken.

**Beachten Sie:** Falls es bei Ihnen einfach nicht klappen will mit dem Schützen von Verzeichnissen, dann könnte der Grund sein, dass in der zentralen Konfiguration des Apache Web-Servers beim Eintrag `AllowOverride` zu wenig erlaubt ist. Falls Sie selber Zugriff auf die Web-Server-Konfiguration haben, suchen Sie dort nach diesem Eintrag und ändern Sie ihn. Weisen Sie z.B. `All` als Wert zu. Fragen Sie ansonsten Ihren Provider oder Webmaster!

Schutzmechanismen, die Sie mit Hilfe von *.htaccess*-Dateien erstellen, sind auf HTTP-Ebene wesentlich sicherer als solche, die mit Hilfe von CGI-Scripts oder gar mit Hilfe von JavaScript erstellt werden. Dennoch sollten Sie wissen, dass *.htaccess* keinen Generalschutz bietet. Der Schutz gilt nur, wenn Web-Browser oder andere Web-Clients über den Web-Server geschützte Daten anfordern. Er gilt nicht, wenn der Zugriff z.B. mit einem anderen Internet-Protokoll wie FTP erfolgt.

### ***IPs, IP-Bereiche oder Namensadressen zulassen/ausschließen***

Sie können bestimmte IP-Adressen oder IP-Bereiche davon ausschließen, auf Web-Seiten zuzugreifen. Ebenso können Sie alle IP-Adressen ausschließen und nur ganz bestimmten den Zugriff erlauben. Anwender, die über eine nicht autorisierte IP-Adresse zugreifen, erhalten dann eine HTTP-Fehlermeldung (den HTTP-Statuscode 403), und der Zugriff wird ihnen verweigert.

Sinnvoll ist das beispielsweise, wenn Ihr Web-Angebot nur den Mitarbeitern Ihrer Firma zugänglich sein soll oder wenn Sie bestimmte Anwender, die mit festen IP-Adressen unterwegs sind, wegen Fehlverhaltens etwa in einem von Ihnen angebotenen, webbasierten Forum oder Chat ausschließen möchten.

Beispiel: *.htaccess*-Datei

```
# Datei zum Regeln von IP-Bereichen
Order deny,allow
Deny from .aol.com
Deny from 192.168
Allow from 192.168.220.102
```

Zunächst legen Sie mit *Order* die logische Interpretationsreihenfolge der nachfolgenden Angaben fest. Möglich ist die Angabe *deny,allow* wie im Beispiel, oder auch die umgekehrte Reihenfolge.

In Zeilen, die mit *Deny from* oder *Allow from* beginnen, geben Sie eine konkrete IP-Adresse, einen Teil davon, eine Namensadresse oder einen Teil davon an. Mit *Deny from* verbieten Sie den Zugriff für den oder die angegebenen Benutzer, und mit *Allow from* erlauben Sie den Zugriff. Per zentraler Voreinstellung ist normalerweise der Zugriff für alle Benutzer erlaubt. In *.htaccess*-Dateien ist es deshalb sinnvoll, vor allem einschränkende Verbote zu formulieren. Im obigen Beispiel werden alle Benutzer ausgesperrt, die mit einer AOL-Kennung surfen, (*.aol.com*), sowie alle Benutzer mit numerischer IP des Bereichs 192.168. Um aber einem bestimmten Benutzer aus diesem Bereich doch den Zugriff zu erlauben, wird im Beispiel anschließend noch mit *Allow from* dessen IP-Adresse angegeben.

Die Angabe wird einfach als Teilzeichenkette interpretiert. Wenn ein Client eine Web-Seite in dem Verzeichnis mit der *.htaccess*-Datei aufruft, vergleicht der Web-Server, ob eine der notierten Zeichenketten in der Zeichenkette vorkommt, die der aufrufende Client dem Server übermittelt. Um mehr über die möglichen Zeichenketten zu erfahren, die in der Praxis übermittelt werden, empfiehlt sich ein Blick in die Log-Dateien des Web-Servers.

**Beachten Sie:** Anstelle einer bestimmten Zeichenkette können Sie auch *Allow from all* bzw. *Deny from all* notieren, um eine generelle Erlaubnis bzw. ein generelles Verbot zu formulieren. Beim Aussperren und Einschließen bestimmter IPs, IP-Bereiche oder Namensadressen sind die gleichen erweiterten Möglichkeiten erlaubt wie beim Passwortschutz. Die Anweisungen mit

Deny from und Allow from können dabei in entsprechende Tags eingeschlossen werden. So lässt sich z.B. das Aussperren bestimmter Benutzer oder Benutzerkreise auf bestimmte Dateien oder Zugriffsmethoden beschränken.

### **Verzeichnis-Optionen einstellen**

Mit Hilfe einer *.htaccess*-Datei können Sie alle zentralen Verzeichnisoptionen, die in der Konfiguration des Web-Servers eingestellt sind, für das entsprechende Verzeichnis und seine Unterverzeichnisse ändern. Voraussetzung ist allerdings, dass in der zentralen Web-Server-Konfiguration AllowOverride All eingetragen ist. Wenn nicht, dann werden Angaben für andere Verzeichnisoptionen ignoriert.

Beispiel: *.htaccess*-Datei:

```
# Datei fuer Verzeichnis /bilder
Options +ExecCGI -Indexes
DirectoryIndex erste.htm start.htm
ErrorDocument 403 "Auf dieses Verzeichnis ist der Zugriff verboten".
ErrorDocument 404 /spezial/404.html
ErrorDocument 500 http://www.mein-anderer-server.de/spezial/500.html
```

Mit Options können Sie verschiedene wichtige Grundeinstellungen ändern, die der Web-Server für Verzeichnisse verwendet. Hinter Options können Sie eine oder mehrere der folgenden Angaben notieren:

- +ExecCGI erlaubt das Ausführen von CGI-Skripts im Verzeichnis, falls es zentral nur im definierten CGI-Verzeichnis erlaubt ist, und -ExecCGI verhindert es, sofern es zentral in allen Verzeichnissen erlaubt ist.
- +Includes erlaubt das Ausführen von Server Side Includes im Verzeichnis, falls es zentral verboten ist, und -Includes verhindert es, sofern es zentral erlaubt ist.
- +IncludesNOEXEC erlaubt das Ausführen von Server Side Includes, die kein CGI-Skript ausführen, -IncludesNOEXEC verbietet solche Server Side Includes.
- +Indexes erlaubt Verzeichnis-Browsing, falls es zentral verboten ist, und -Indexes verhindert es, sofern es zentral erlaubt ist. Beim Verzeichnis-Browsing wird der Inhalt des Verzeichnisses im Browser aufgelistet, falls nur das Verzeichnis aufgerufen wird (z.B. <http://www.ihr-name.de/bilder/>) und dort keine Datei mit einem Namen zu finden ist, der »Index-Funktion« hat. Mit +MultiViews erlauben Sie das Definieren alternativer Inhalte, mit -MultiViews schalten Sie es aus.

Im Normalfall ist der Web-Server so konfiguriert, dass er das »Stöbern« im Verzeichnis zulässt, falls dort keine Datei mit »Index-Funktion« existiert. Das sind in der Einstellung der meisten Web-Server Dateien mit Namen wie *index.htm* oder *index.html*. Mit DirectoryIndex können Sie andere Dateinamen bestimmen, die in diesem Verzeichnis und seinen Unterverzeichnissen als Dateien mit Index-Funktion gelten. Sie können beliebig viele Dateinamen angeben.

Per Voreinstellung zeigt der Web-Server eine in seinen Konfigurationen enthaltene HTML-Datei an, wenn ein Zugriffsfehler passiert, z.B. wenn eine angeforderte Datei nicht existiert. Auch dies können Sie ändern: Mit ErrorDocument definieren Sie eine eigene Meldung oder Datei, die im Fehlerfall angezeigt wird. Hinter dem Schlüsselwort notieren Sie den gewünschten HTTP-Status-

code – typische Fehler, die auftreten können, sind jene mit den Nummern 403, 404 und 500. Hinter der Nummer mit dem Statuscode können Sie entweder in Anführungszeichen eine eigene Fehlermeldung notieren, oder Sie geben die Adresse einer Web-Seite an, die im Fehlerfall angezeigt werden soll. Bei Adressen auf der eigenen Domain können Sie mit absoluten Pfadnamen arbeiten, wobei das Wurzelverzeichnis des Web-Projekts der Ausgangspunkt ist. Falls Sie nur eine Fehlermeldung notieren, wird diese bei Eintreten des Fehlers als reiner Text im Browser angezeigt. Angegebene Web-Seiten können Sie nach Lust und Laune gestalten. Es ist beispielsweise für Besucher angenehm und weniger erschreckend, wenn im Fall eines Fehlers eine Seite angezeigt wird, die das gleiche Layout hat wie die übrigen Projektseiten.

### Verzeichnis-Browsing einstellen

Wenn Sie Verzeichnis-Browsing erlauben, sodass also bei einem Aufruf wie `http://www.ihr-name.de/bilder/` der Verzeichnisinhalt aufgelistet wird, können Sie die Optik beeinflussen, mit der die Verzeichnislisten im Browser angezeigt werden. Leichter verständlich wird das, wenn Sie sich klarmachen, dass der Web-Browser beim Anzeigen eines Verzeichnis-Listings einfach dynamisch HTML-Code generiert, um den eingelesenen Verzeichnisinhalt anzuzeigen. Mit den hier beschriebenen Optionen können Sie die HTML-Ausgabe des Web-Servers beeinflussen.

| Index of /bilder                                                                                                     |                               |                      |                             |  |
|----------------------------------------------------------------------------------------------------------------------|-------------------------------|----------------------|-----------------------------|--|
|  <a href="#">Name</a>              | <a href="#">Last modified</a> | <a href="#">Size</a> | <a href="#">Description</a> |  |
|  <a href="#">Parent Directory</a> | 22-Dec-2000 18:35             | -                    |                             |  |
|  <a href="#">bild1.gif</a>        | 27-Apr-1998 07:00             | 5k                   | GIF-Grafik, anzeigbar       |  |
|  <a href="#">bild2.gif</a>        | 27-Apr-1998 07:00             | 6k                   | GIF-Grafik, anzeigbar       |  |
|  <a href="#">bild3.gif</a>        | 27-Apr-1998 07:00             | 6k                   | GIF-Grafik, anzeigbar       |  |
|  <a href="#">blau1.gif</a>        | 27-Jul-2001 10:19             | 1k                   | GIF-Grafik, anzeigbar       |  |
|  <a href="#">blue.gif</a>         | 24-Jul-2001 16:00             | 1k                   | GIF-Grafik, anzeigbar       |  |
|  <a href="#">breeze.mid</a>       | 27-Apr-1998 07:00             | 17k                  |                             |  |
|  <a href="#">datei2.htm</a>       | 29-Jun-2001 22:05             | 1k                   | HTML-Datei, anzeigbar       |  |
|  <a href="#">denker.jpg</a>       | 27-Apr-1998 07:00             | 9k                   | JPEG-Grafik, anzeigbar      |  |

A.1: Die Abbildung zeigt, wie ein Browser ein Verzeichnis-Listing anzeigen könnte, bei dem eigene Dateibesreibungen und Symbolgrafiken zum Einsatz kommen.

Beispiel: `.htaccess`-Datei

```
# Datei fuer Verzeichnis /bilder
FancyIndexing On
AddDescription "HTML-Datei, anzeigbar" *.htm *.html
AddDescription "GIF-Grafik, anzeigbar" *.gif
AddDescription "JPEG-Grafik, anzeigbar" *.jpg
AddDescription "ZIP-Archiv, downloadbar" *.zip
AddIcon /src/blau.gif html htm
AddIcon /src/gelb.gif gif jpg
```



```
AddIcon /src/gruen.gif ^^DIRECTORY^^
AddIcon /src/blau.gif ^^BLANKICON^^
```

Mit `FancyIndexing On` schalten Sie eigene Optionen zur Darstellung des Verzeichnis-Listings ein. Die Einträge der übrigen Zeilen im obigen Beispiel definieren solche Anzeigeeoptionen. Mit `AddDescription` können Sie einen kurzen Beschreibungstext notieren, der hinter einer oder mehreren bestimmten Dateien stehen soll. Zuerst notieren Sie in Anführungszeichen den gewünschten Text, und dahinter geben Sie an, für welche Datei oder Dateien der Text gelten soll. Dabei können Sie wie im Beispiel oben Wildcards benutzen.

Mit `AddIcon` können Sie für eine oder mehrere bestimmte Dateien eine eigene Symbolgrafik bestimmen. Dabei geben Sie den URI der Symbolgrafik an. Bei Adressen auf der eigenen Domain können Sie mit absoluten Pfadnamen arbeiten, wobei das Wurzelverzeichnis des Web-Projekts der Ausgangspunkt ist. Hinter der Angabe zur gewünschten Grafikdatei können Sie eine oder mehrere Dateiendungen notieren. Dateien mit dieser Endung erhalten dann bei der Anzeige die entsprechende Symbolgrafik. Anstelle von Dateiendungen können Sie auch reservierte Ausdrücke wie `^^DIRECTORY^^` (Verzeichnisse erhalten diese Symbolgrafik) oder `^^BLANKICON^^` (Dateien ohne Dateiendung oder nicht zuzuordnende Dateien erhalten diese Symbolgrafik) notieren.

Weitere Optionen sind ebenfalls möglich, aber im obigen Beispiel nicht notiert:

- Mit `AddType` können Sie einem Mime-Typ eine oder mehrere Dateiendungen zuordnen: Mit `AddType text/xml adr` ordnen Sie beispielsweise Dateien mit `*.adr` dem Mime-Type `text/xml` zu.
- Mit `AddEncoding` können Sie dem Sub-Typ eines Mime-Typs eine oder mehrere Dateiendungen zuordnen: Mit `AddEncoding javascript jscr` ordnen Sie beispielsweise Dateien mit `*.jsr` dem Mime-Type `text/javascript` zu.
- Mit `AddIconByType` können Sie einem Mime-Typ eine Symbolgrafik zuordnen. Alle Dateien dieses Mime-Typs werden dann mit dieser Symbolgrafik angezeigt. Mit `AddIconByType text/xml /src/struktur.gif` ordnen Sie beispielsweise Dateien mit dem Mime-Type `text/xml` eine Grafik mit der Adresse `/src/struktur.gif` zu.
- Mit `AddIconByEncoding` können Sie dem Sub-Typ eines Mime-Typs eine Symbolgrafik zuordnen: Mit `AddIconByEncoding javascript /src/action.gif` ordnen Sie beispielsweise Dateien mit dem Mime-Subtyp `javascript` eine Grafik mit der Adresse `/src/action.gif` zu.
- Mit `DefaultIcon` bestimmen Sie eine Symbolgrafik für alle Fälle, die nicht durch speziellere Angaben abgedeckt sind: Mit `DefaultIcon /src/punkt.gif` bestimmen Sie beispielsweise, dass alle Verzeichniseinträge, die sich nicht anderweitig zu Symbolgrafiken zuordnen lassen, die Grafik mit der Adresse `/src/punkt.gif` erhalten.
- Mit `HeaderName` bestimmen Sie eine eigene Datei für die Anzeige der Kopfinformation zum Verzeichnis: Mit `HeaderName kopf.htm` bestimmen Sie beispielsweise, dass die Anzeige der Kopfinformationen aus der Datei `kopf.htm` im gleichen Verzeichnis geholt werden.
- Mit `ReadmeName` bestimmen Sie eine Datei, in der zusätzliche Informationen zum Verzeichnis stehen, z.B. Copyright-Hinweise. Diese Datei wird unterhalb der Verzeichnisliste eingebun-

den. Mit `HeaderName fuss.htm` bestimmen Sie beispielsweise, dass die Anzeige der »Liesmich«-Informationen aus der Datei `fuss.htm` im gleichen Verzeichnis geholt wird.

- Mit `IndexIgnore` können Sie Dateien angeben, die nicht im Verzeichnis-Listing auftauchen sollen: Mit `IndexIgnore kopf.htm fuss.htm nacktbild.jpg *.bmp` bestimmen Sie beispielsweise, dass die Dateien `kopf.htm`, `fuss.htm`, `nacktbild.jpg` sowie alle Dateien mit der Endung `.bmp` nicht angezeigt werden.
- Mit `IndexOrderDefault` wird die Sortierung bestimmt. Per Voreinstellung werden die Verzeichniseinträge alphabetisch aufsteigend nach Dateinamen angezeigt. Mit `IndexOrderDefault Descending Name` können Sie beispielsweise nach Dateinamen absteigend sortieren, mit `IndexOrderDefault Ascending Date` aufsteigend nach Zeitstempel, mit `IndexOrderDefault Descending Size` absteigend nach Dateigröße und mit `IndexOrderDefault Ascending Description` aufsteigend nach Dateikurzbeschreibung.

- Mit `AddAlt` können Sie einer verwendeten Symbolgrafik einen HTML-Alternativtext zuordnen, falls die Grafik nicht angezeigt werden kann: Mit `AddAlt "das ist eine Grafikdatei" gelb.gif` bestimmen Sie beispielsweise den Alternativtext `das ist eine Grafikdatei` für die Symbolgrafik `gelb.gif`.

Mit `AddAltByType` können Sie allen Symbolgrafiken vor Dateien eines bestimmten Mime-Typs einen gemeinsamen HTML-Alternativtext zuordnen. Mit `AddAltByType "Das ist eine XML-basierte Datei" text/xml` ordnen Sie beispielsweise allen Symbolgrafiken, die vor einer Datei mit dem Mime-Type `text/xml` stehen, den angegebenen Alternativtext zu. Mit `AddAltByEncoding` können Sie allen Symbolgrafiken vor Dateien eines bestimmten Mime-Subtyps einen gemeinsamen HTML-Alternativtext zuordnen.

Mit `AddAltByEncoding "Das ist eine XML-basierte Datei" javascript` ordnen Sie beispielsweise allen Symbolgrafiken, die vor einer Datei mit dem Mime-Subtyp `javascript` stehen, den angegebenen Alternativtext zu.

- Mit `IndexOptions` können Sie schließlich weitere Einzeloptionen festlegen: Mit `IndexOptions IconHeight=12 IconWidth=14 IconsAreLinks ScanHTMLTitles` legen Sie beispielsweise fest, dass Icon-Grafiken in der fixen Höhe von 12 Pixeln und der fixen Breite von 14 Pixeln angezeigt werden. Die Icons sind außerdem als Links anklickbar und rufen die dahinter angezeigte Datei oder das Verzeichnis auf. In der Spalte für Dateibeschreibungen werden im Fall von HTML-Dateien die Inhalte der darin definierten `title`-Elemente ausgelesen und als Kurzbeschreibung zur Datei angezeigt.

Weitere Optionen sind:

Mit `SuppressColumnSorting` werden die Verzeichniseinträge unsortiert angezeigt. Mit `SuppressDescription` werden keine Kurzbeschreibungen angezeigt. Mit `SuppressLastModified` werden keine Zeitstempel bei den Einträgen angezeigt. Mit `SuppressSize` werden keine Dateigrößen angezeigt.

### Alternative Inhalte senden

Sie können die Kommunikation zwischen Web-Server und aufrufendem Client beeinflussen und dafür sorgen, dass je nach Client-Typ verschiedene Inhalte gesendet werden. Auf diese Weise können Sie beispielsweise eine deutschsprachige oder englischsprachige Index-Seite senden, abhängig davon, ob der Anwender einen Browser mit deutschsprachiger oder englischsprachiger Benutzeroberfläche benutzt. Oder Sie können alternativ eine HTML-Datei oder eine WML-Datei

senden, abhängig davon, ob der aufrufende Client ein normaler Web-Client ist oder ein Handy, das WAP/WML-basierte Inhalte haben möchte.

Um alternative Inhalte anzubieten, sind ein paar Einstellungen in der *.htaccess* nötig (sofern nicht schon in der zentralen Web-Server-Konfiguration eingestellt). Außerdem benötigen Sie eine »Verteilerdatei«.

Beispiel: *.htaccess*-Datei

```
# Einstellungen fuer alternative Inhalte
AddType application/x-type-map var
Options +MultiViews
DirectoryIndex index
```

Beispiel: Datei *index.var*

```
URI: index; vary="type,language"

URI: english.htm
Content-type: text/html; qs=1.0
Content-language: en

URI: deutsch.htm
Content-type: text/html; qs=0.8
Content-language: de
```

In der *.htaccess* stehen folgende Einträge: `AddType application/x-type-map var` fügt einen Mime-Type hinzu, den der Web-Server (zumindest Apache) kennt. Es ist der Mime-Type für Dateien mit Angaben zu alternativen Inhalten. Als Dateierweiterung für solche Dateien wird im Beispiel `var` definiert. Dateien `*.var` werden also als Anweisungsdateien für alternative Inhalte erkannt. `Options +MultiViews` schaltet die MultiView-Funktion ein. Für alternative Inhalte muss diese Option explizit eingeschaltet sein. `DirectoryIndex index` nennt Dateien, in deren Namen `index` vorkommt, als Dateien für die Index-Funktion.

Als Nächstes benötigen Sie im Beispiel eine Datei namens *index.var* im gleichen Verzeichnis wie die *.htaccess*. Die Datei *index.var* enthält im Beispiel in der ersten Zeile die folgende Angabe:

```
URI: index; vary="type,language"
```

Damit drücken Sie aus, dass die folgenden Definitionen für alternativen Inhalt dann berücksichtigt werden sollen, wenn Anfragen kommen, die auf den festgelegten `DirectoryIndex` (im Beispiel zuvor als `index` definiert) zutreffen. Die Alternativen können Angaben zum Dateityp (`type`) und zur Sprache (`language`) betreffen.

Diese erste Anweisung und die möglichen Alternativen werden jeweils durch eine Leerzeile getrennt. Jede Alternative beginnt mit `URI:`. Dahinter folgt der gewünschte URI, der bei der jeweiligen Alternative aufgerufen werden soll. Im obigen Beispiel handelt es sich jeweils um Dateien im gleichen Verzeichnis, nämlich einmal `english.htm` und einmal `deutsch.htm`.

Mit `Content-type:` bestimmen Sie den Mime-Type des bei `URI:` angegebenen URIs, im obigen Beispiel `text/html`. Falls Sie z.B. alternative Inhalte für Web-Browser und WAP-Handys senden wollen, können Sie bei dieser Angabe in der einen Alternative beispielsweise eine WML-Datei anbieten mit dem Mime-Type `text/vnd.wap.wml`, und im anderen Fall eine HTML-Datei mit

dem üblichen Mime-Type `text/html`. Die Angabe `Content-type:` ist in jedem Fall Pflicht, auch dann, wenn Sie dort keine unterschiedlichen Angaben zu machen haben.

Bei `Content-type` können Sie am Ende, durch Semikolon getrennt, noch eine Angabe zu `qs` notieren. Hinter diesem Schlüsselwort und einem Gleichheitszeichen können Sie numerische Werte für die Alternativen vergeben. Alternativen mit höheren Werten werden in dem Fall, dass sonst nicht entscheidbar ist, welche Alternative gewählt werden soll, bevorzugt.

Mit `Content-language:` können Sie Sprachen unterscheiden, indem Sie ein Sprachenkürzel angeben. Abhängig davon, welche Sprache der aufrufende Client benutzt oder bevorzugt, wird die entsprechende Alternative ausgewählt. Für alternative Inhalte benötigen Sie natürlich auch alternative Dateien. Im obigen Beispiel wären das die Dateien `english.htm` und `deutsch.htm`, die den gleichen Inhalt haben, nur in verschiedenen Sprachen.

Wenn Sie eine solche `.htaccess` und `index.var` ins Startverzeichnis einer Domain legen, z.B. ins Startverzeichnis von `http://www.ihr-name.de/`, dann werden Besucher, die nur die Domain aufrufen (ohne bestimmte Dateinamen), automatisch auf die softwareseitig erkennbare Landessprachenseite geleitet. Von dort aus können entsprechend unterschiedliche Verlinkungen so durch das restliche Angebot führen, dass die Landessprache die gleiche bleibt. Gehen Sie mit dieser Möglichkeit der Lenkung jedoch bedächtig um. Bedenken Sie, dass nicht jeder Besucher Ihres Web-Projekts frei in der Wahl seines Browsers ist. Ein deutschsprachiger Besucher in einem englischen Internet-Cafe hat auch bei besuchten spanischen Seiten eigene Vorstellungen bei der Wahl der Sprache. Sehen Sie gegebenenfalls eine clientseitige Auswahl der Sprache vor.

## B HTML: Elementreferenz

### B.1 Allgemeines zur HTML-Elementreferenz

Ein HTML-Element besteht meistens aus einem Anfangs- und einem End-Tag. So besteht z.B. das Element `body` aus einem Anfangs-Tag `<body>` und einem End-Tag `</body>`. Der Bereich zwischen Anfangs- und End-Tag nimmt den Inhalt des Elements auf. Er kann weitere HTML-Elemente enthalten.

Die HTML-Elementreferenz listet für jedes HTML-Element auf, in welchen anderen HTML-Elementen es vorkommen kann und welche anderen HTML-Elemente es enthalten kann. Bei jedem aufgeführten Element führt ein Verweis zu den jeweiligen Attributen des Elements, die in der Attributreferenz behandelt werden.

In der HTML-Elementreferenz wird nur der HTML-Standard nach Version 4.01 berücksichtigt. Browser-eigene Abweichungen sowie Browser-spezifische HTML-Elemente, die im Standard nicht vorkommen, sind in der Referenz nicht berücksichtigt.

### B.2 Block-Elemente

Block-Elemente erzeugen eine neue Zeile im Textfluss. Je nach Elementtyp kann der Browser dabei automatisch einen größeren Abstand zum vorangehenden Inhalt einfügen. So erzeugen die Browser beispielsweise beim `p`-Element einen sichtbaren Abstand, damit die Absatzschaltung vom einfachen Zeilenumbruch unterscheidbar ist. Beim `div`-Element dagegen wird beispielsweise einfach eine neue Zeile begonnen.

Block-Elemente können in der Regel normalen Text und Inline-Elemente enthalten. Einige Block-Elemente können auch andere Block-Elemente enthalten. Ausnahmen von der Regel werden bei den Beschreibungen zu den einzelnen Elementen behandelt.

Folgende Elemente gehören zu den Block-Elementen: `address`, `blockquote`, `center`, `dl`, `div`, `div`, `fieldset`, `form`, `h1-6`, `hr`, `isindex`, `menu`, `noframes`, `noscript`, `ol`, `p`, `pre`, `table`, `ul`. Nur bei HTML transitional gibt es `center`, `div`, `menu`, `noframes` und `isindex`.

### B.3 Inline-Elemente

Inline-Elemente erzeugen **keine** neue Zeile im Textfluss. Inline-Elemente sind als untergeordnete, »innere« Elemente für Block-Elemente gedacht. Sie können in den meisten Fällen normalen Text und weitere Inline-Elemente enthalten, jedoch keine Block-Elemente. Ausnahmen oder Einschränkungen werden bei den Beschreibungen zu den einzelnen Elementen behandelt.

Folgende Elemente gehören zu den **Inline-Elementen**: a, abbr, acronym, applet, b, basefont, bdo, big, br, button, cite, code, dfn, em, font, i, img, input, iframe, kbd, label, map, object, q, samp, script, select, small, span, strong, sub, sup, textarea, tt, var. Nur bei HTML transitional gibt es applet, basefont, font, iframe, u, s und strike.

## B.4 #PCDATA

*#PCDATA* steht für *parsed character data*, zu deutsch: vom HTML-Parser analysierte Zeichendaten. Es handelt sich in HTML normalerweise um Text, der innerhalb eines Elements, also zwischen Anfangs- und End-Tag notiert werden kann. Solcher Text kann aus Zeichen bestehen, die in der Zeichenreferenz genannt sind. Benannte Zeichen wie &uuml; sind also ebenso erlaubt wie numerische Notationen von der Art &#x20AC; oder &#8364;. Solche Notationen werden erkannt und umgesetzt. Im Unterschied zu *#CDATA*-Bereichen werden HTML-Auszeichnungen in *#PCDATA*-Bereichen erkannt und interpretiert.

## B.5 Elementreferenz

### **a**

|                |                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert einen Anker oder einen Hyperlink.                                                                                                         |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                            |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer a, button), body (body nur bei HTML transitional). |
| Kindelemente   | Darf <i>#PCDATA</i> und folgende andere HTML-Elemente enthalten: [Inline-Elemente] (außer a).                                                       |

### **abbr**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Abkürzung.                                                                                                          |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf <i>#PCDATA</i> und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                               |

### **acronym**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Abkürzung.                                                                                                          |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf <i>#PCDATA</i> und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                               |

**address**

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Adresse.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                                                                                                                                                                                                                                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: <code>applet</code> , <code>blockquote</code> , <code>body</code> , <code>button</code> , <code>center</code> , <code>dd</code> , <code>del</code> , <code>div</code> , <code>fieldset</code> , <code>form</code> , <code>iframe</code> , <code>ins</code> , <code>li</code> , <code>map</code> , <code>noframes</code> , <code>noscript</code> , <code>object</code> , <code>td</code> , <code>th</code> . |
| Kindelemente   | Darf <code>#PCDATA</code> und folgende andere HTML-Elemente enthalten: [Inline-Elemente], <code>p</code> ( <code>p</code> nur bei HTML transitional).                                                                                                                                                                                                                                                                                                             |

**applet**

|                |                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Fügt ein Java-Applet ein.                                                                                                                                                              |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                                       |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer <code>pre</code> ), [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf folgende anderen HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente], <code>param</code> .                                                                               |

**area**

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| Bedeutung      | Definiert einen Ausschnitt für eine verweissensitive Grafik.                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                     |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>map</code> . |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                   |

**b**

|                |                                                                                                                                                              |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert fett gedruckten Text.                                                                                                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                     |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf <code>#PCDATA</code> und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                    |

**base**

|                |                                                                                      |
|----------------|--------------------------------------------------------------------------------------|
| Bedeutung      | Definiert dokumentweit gültige Angaben für Zielfenster und Basis-URIs von Verweisen. |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                             |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>head</code> .        |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                           |

***basefont***

|                |                                                                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Bestimmt Basis-Schriftmerkmale für nachfolgenden Text.                                                                                        |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                              |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer pre), [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                                                                                    |

***bdo***

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Änderung der Textrichtung bei Sprachen mit anderer Schreibrichtung.                                                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

***big***

|                |                                                                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text, der größer dargestellt werden soll als normal.                                                                                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                      |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer pre), [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                  |

***blockquote***

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert ein Zitat.                                                                                                                                                                     |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten:<br>1. nach HTML strict: [Block-Elemente], script<br>2. nach HTML transitional: [Block-Elemente], [Inline-Elemente].           |

***body***

|                |                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert den Dokumentkörper und damit den Bereich der darzustellenden Daten.                                                                    |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                        |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: html, noframes (noframes nur bei HTML frameset).                                          |
| Kindelemente   | Darf enthalten:<br>1. nach HTML strict: [Block-Elemente], script<br>2. nach HTML transitional: #PCDATA und [Block-Elemente], [Inline-Elemente]. |



***br***

|                |                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Erzeugt einen Zeilenumbruch.                                                                                                                     |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                         |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <i>body</i> ( <i>body</i> nur bei HTML transitional). |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                                                                                       |

***button***

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Erzeugt einen Klick-Button.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer <i>button</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: <i>abbr</i> , <i>acronym</i> , <i>address</i> , <i>applet</i> , <i>b</i> , <i>basefont</i> , <i>bdo</i> , <i>big</i> , <i>blockquote</i> , <i>br</i> , <i>center</i> , <i>cite</i> , <i>code</i> , <i>dfn</i> , <i>dl</i> , <i>dir</i> , <i>div</i> , <i>em</i> , <i>font</i> , <i>h1-6</i> , <i>hr</i> , <i>i</i> , <i>img</i> , <i>kbd</i> , <i>map</i> , <i>menu</i> , <i>noframes</i> , <i>noscript</i> , <i>object</i> , <i>ol</i> , <i>p</i> , <i>pre</i> , <i>q</i> , <i>samp</i> , <i>script</i> , <i>small</i> , <i>span</i> , <i>strong</i> , <i>sub</i> , <i>sup</i> , <i>table</i> , <i>tt</i> , <i>ul</i> , <i>var</i> . |

***caption***

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| Bedeutung      | Setzt eine Tabellenüberschrift/Tabellenunterschrift.                         |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                     |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <i>table</i> .     |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente]. |

***center***

|                |                                                                                                                                                                                                                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Bereich, der zentriert dargestellt wird.                                                                                                                                                                                                                                                                                         |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                                                                                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: <i>applet</i> , <i>blockquote</i> , <i>body</i> , <i>button</i> , <i>center</i> , <i>dd</i> , <i>del</i> , <i>div</i> , <i>fieldset</i> , <i>form</i> , <i>iframe</i> , <i>ins</i> , <i>li</i> , <i>map</i> , <i>noframes</i> , <i>noscript</i> , <i>object</i> , <i>td</i> , <i>th</i> . |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente].                                                                                                                                                                                                                                                  |

***cite***

|                |                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert ein Zitat.                                                                                                                              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                         |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <i>body</i> ( <i>body</i> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                     |

**code**

|                |                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als Quelltext einer Computersprache.                                                                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                         |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <i>body</i> ( <i>body</i> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                     |

**col**

|                |                                                                                        |
|----------------|----------------------------------------------------------------------------------------|
| Bedeutung      | Definiert Daten für eine Tabellenspalte.                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                               |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: <i>colgroup</i> , <i>table</i> . |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                             |

**colgroup**

|                |                                                                                        |
|----------------|----------------------------------------------------------------------------------------|
| Bedeutung      | Definiert eine Gruppe von Tabellenspalten.                                             |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                               |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <i>table</i> .               |
| Kindelemente   | Darf nur folgendes andere HTML-Element (einfach oder mehrfach) enthalten: <i>col</i> . |

**dd**

|                |                                                                                                |
|----------------|------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert in einer Definitionsliste eine Definition zu einem Term.                              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                       |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <i>dl</i> .                          |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente]. |

**del**

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als gelöscht im Sinne von Änderungsmarkierung.                                             |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                 |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <i>body</i> . |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente].           |

**dfn**

|                |                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Definition.                                                                                                                        |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                         |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <i>body</i> ( <i>body</i> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                     |

***dir***

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Verzeichnisliste.                                                                                                                                                         |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                                        |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf folgendes andere HTML-Element enthalten (ein- oder mehrmals): li (Hinweis: li darf in diesem Zusammenhang keine Block-Elemente enthalten).                                         |

***div***

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen allgemeinen Bereich.                                                                                                                                                     |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente].                                                                                          |

***dl***

|                |                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Definitionsliste.                                                                                                                                                        |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                               |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th |
| Kindelemente   | Darf folgende andere HTML-Elemente enthalten: dd, dt.                                                                                                                                  |

***dt***

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| Bedeutung      | Markiert in einer Definitionsliste einen zu definierenden Term.              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                     |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: dl.                |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente]. |

***em***

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als emphatisch.                                                                                                      |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                           |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional) . |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                       |

**fieldset**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Gruppe von Formularelementen.                                                                                                                                             |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf neben #PCDATA folgende HTML-Elemente enthalten: legend, gefolgt von: [Block-Elemente], [Inline-Elemente].                                                                          |

**font**

|                |                                                                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text mit Schriftgröße, Schriftfarbe und Schriftart.                                                                                  |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                              |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer pre), [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                  |

**form**

|                |                                                                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert ein Formular.                                                                                                                                                           |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | 1. nach HTML strict: [Block-Elemente] (außer form), script<br>2. nach HTML transitional: [Block-Elemente] (außer form), [Inline-Elemente].                                        |

**frame**

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| Bedeutung      | Definiert ein Frame-Fenster.                                        |
| HTML-Varianten | Erlaubt in der Variante frameset.                                   |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: frameset. |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.          |

**frameset**

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| Bedeutung      | Definiert ein Formular.                                           |
| HTML-Varianten | Erlaubt in der Variante frameset.                                 |
| Elternelemente | Darf innerhalb des folgenden HTML-Elements vorkommen: html.       |
| Kindelemente   | Darf folgende HTML-Elemente enthalten: frame, frameset, noframes. |

**h1-h6**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Überschrift 1. bis 6. Ordnung.                                                                                                                                            |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                                            |

**head**

|                |                                                                                                                                                         |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert den Kopfbereich einer HTML-Datei.                                                                                                              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                |
| Elternelemente | Darf innerhalb des folgenden HTML-Elements vorkommen: html.                                                                                             |
| Kindelemente   | Muss folgendes HTML-Element enthalten: title<br>Kann darüber hinaus folgende HTML-Elemente enthalten: base, isindex, link, meta, object, script, style. |

**hr**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Erzeugt eine horizontale Trennlinie.                                                                                                                                                    |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                                                                                                                              |

**html**

|                |                                                                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Wurzelement einer HTML-Datei.                                                                                                                                         |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                              |
| Elternelemente | Hat keine möglichen Elternelemente.                                                                                                                                   |
| Kindelemente   | 1. nach HTML strict und HTML transitional folgende HTML-Elemente: head, gefolgt von body<br>2. nach HTML frameset folgende HTML-Elemente: head, gefolgt von frameset. |

**i**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert kursiv gedruckten Text – siehe Beschreibung.                                                                             |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

**iframe**

|                |                                                                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert ein eingebettetes Fenster.                                                                                                                                                      |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer <code>button</code> ), <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf <code>#PCDATA</code> und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente].                                                                               |

**img**

|                |                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Referenziert eine Grafik.                                                                                                                                                              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                               |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer <code>pre</code> ), [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                                                                                                                             |

**input**

|                |                                                                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Erzeugt ein Formularelement, Typ wird durch Attribut bestimmt.                                                                                                                            |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                  |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer <code>button</code> ), <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                                                                                                                                |

**ins**

|                |                                                                                                                |
|----------------|----------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als eingefügt im Sinne von Änderungsmarkierung.                                                  |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                       |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <code>body</code> . |
| Kindelemente   | Darf <code>#PCDATA</code> und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente].    |

**isindex**

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Erzeugt ein einzeliges Eingabefeld.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                                                                                                                                                                                                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: <code>applet</code> , <code>blockquote</code> , <code>body</code> , <code>center</code> , <code>dd</code> , <code>del</code> , <code>div</code> , <code>fieldset</code> , <code>form</code> , <code>head</code> , <code>iframe</code> , <code>ins</code> , <code>li</code> , <code>map</code> , <code>noframes</code> , <code>noscript</code> , <code>object</code> , <code>td</code> , <code>th</code> . |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                                                                                                                                                                                                                                                                                                                                                                                                      |

***kbd***

|                |                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als Tastatureingabe.                                                                                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                         |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <i>body</i> ( <i>body</i> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                     |

***label***

|                |                                                                                                                                                                         |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Ordnet Text einem bestimmten Formularfeld zu.                                                                                                                           |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer <i>button</i> ), <i>body</i> ( <i>body</i> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente] (außer <i>label</i> ).                                                                      |

***legend***

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| Bedeutung      | Definiert eine Überschrift für eine Gruppe von Formularelementen.            |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                     |
| Elternelemente | Darf nur innerhalb der folgenden HTML-Elements vorkommen: <i>fieldset</i> .  |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente]. |

***li***

|                |                                                                                                                                                                                             |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Listeneintrag.                                                                                                                                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                    |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: <i>div</i> , <i>menu</i> , <i>ol</i> , <i>ul</i> .                                                                                    |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten:<br>1. bei <i>div</i> und <i>menu</i> : [Inline-Elemente]<br>2. bei <i>ol</i> und <i>ul</i> : [Block-Elemente], [Inline-Elemente]. |

***link***

|                |                                                                         |
|----------------|-------------------------------------------------------------------------|
| Bedeutung      | Definiert logische Beziehungen zu anderen Dateien.                      |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <i>head</i> . |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.              |

**map**

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Definitionsbereich für verweissensitive Grafiken.                           |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                   |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente]. |
| Kindelemente   | Darf folgende anderen HTML-Elemente enthalten: [Block-Elemente], area.                     |

**menu**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Menüliste.                                                                                                                                                                |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                                        |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf folgendes andere HTML-Element enthalten (ein- oder mehrmals): li (Hinweis: li darf in diesem Zusammenhang keine Block-Elemente enthalten).                                         |

**meta**

|                |                                                                           |
|----------------|---------------------------------------------------------------------------|
| Bedeutung      | Definiert Meta-Daten wie Kurzbeschreibung und Stichwörter für eine Datei. |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                  |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: head.           |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.                |

**noframes**

|                |                                                                                                                                                                               |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Bereich für den Fall, dass keine Frames angezeigt werden können.                                                                                               |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                                              |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noscript, object, td, th. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten:<br>1. nach HTML frameset: body<br>2. nach HTML transitional: [Block-Elemente], [Inline-Elemente].                   |

**noscript**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Bereich für den Fall, dass ein Script nicht ausgeführt werden kann.                                                                                                      |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente].                                                                                          |



**object**

|                |                                                                                                              |
|----------------|--------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert ein Multimedia-Objekt.                                                                             |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                     |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer pre), [Inline-Elemente], head. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente], param.        |

**ol**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine nummerierte Liste.                                                                                                                                                        |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf folgendes andere HTML-Element enthalten (ein- oder mehrmals): li.                                                                                                                  |

**optgroup**

|                |                                                                            |
|----------------|----------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Gruppe von Einträgen in einer Menüstruktur.                  |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                   |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: select.          |
| Kindelemente   | Darf folgendes andere HTML-Element enthalten (ein- oder mehrmals): option. |

**option**

|                |                                                                         |
|----------------|-------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Eintrag in einer Auswahlliste.                           |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: select, optgroup. |
| Kindelemente   | Darf nichts als #PCDATA enthalten.                                      |

**p**

|                |                                                                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Textabsatz.                                                                                                                                                                       |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                         |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: address, applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                                                     |

**param**

|                |                                                                       |
|----------------|-----------------------------------------------------------------------|
| Bedeutung      | Definiert Parameter eines Multimedia-Objekts oder Java-Applets.       |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.              |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, object. |
| Kindelemente   | Leeres Element (Standalone-Tag), darf keinen Inhalt haben.            |

**pre**

|                |                                                                                                                                                                                                                           |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Formatiert einen Bereich wie im Editor eingegeben (präformatierter Text).                                                                                                                                                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                                                  |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th.                                   |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: a, abbr, acronym, applet, b, bdo, br, button, cite, code, dfn, em, i, input, iframe, kbd, label, map, q, samp, script, select, span, strong, textarea, tt, var. |

**q**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert ein Zitat.                                                                                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

**s**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als durchgestrichen.                                                                                                |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                  |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

**samp**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text als Beispiel.                                                                                                       |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

**script**

|                |                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert einen Bereich für Scripts (z.B. JavaScript).                                                                                  |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], head, body (body nur bei HTML transitional). |
| Kindelemente   | Darf nichts als #PCDATA enthalten, die als Script-Code zu interpretieren sind.                                                          |

***select***

|                |                                                                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert eine Auswahlliste.                                                                                                                                                              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                  |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer <code>button</code> ), <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: <code>optgroup</code> , <code>option</code> .                                                                                   |

***small***

|                |                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert Text, der kleiner dargestellt wird als normal.                                                                                                                                |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                               |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer <code>pre</code> ), [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                                           |

***span***

|                |                                                                                                                                                              |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen inline-Bereich zur Formatierung, z.B. mit CSS-Stylesheets.                                                                                    |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                     |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                 |

***strike***

|                |                                                                                                                                                              |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text als durchgestrichen.                                                                                                                     |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                                             |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                 |

***strong***

|                |                                                                                                                                                              |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text als betont.                                                                                                                              |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                     |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                 |

**style**

|                |                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Bereich für Stylesheet-Definitionen, z.B. in CSS.                                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                         |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>head</code> .                    |
| Kindelemente   | Darf nichts als <code>#PCDATA</code> enthalten, die als Style-Definitionen interpretiert werden. |

**sub**

|                |                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text als tiefgestellt.                                                                                                                                                  |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                               |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer <code>pre</code> ), [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf <code>#PCDATA</code> und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                              |

**sup**

|                |                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text als hochgestellt.                                                                                                                                                  |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                               |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente] (außer <code>pre</code> ), [Inline-Elemente], <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf <code>#PCDATA</code> und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                                                              |

**table**

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Erzeugt eine Tabelle.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: <code>applet</code> , <code>blockquote</code> , <code>body</code> , <code>button</code> , <code>center</code> , <code>dd</code> , <code>del</code> , <code>div</code> , <code>fieldset</code> , <code>form</code> , <code>iframe</code> , <code>ins</code> , <code>li</code> , <code>map</code> , <code>noframes</code> , <code>noscript</code> , <code>object</code> , <code>td</code> , <code>th</code> .               |
| Kindelemente   | Darf folgende anderen HTML-Elemente (in dieser Reihenfolge) enthalten: <code>caption</code> (optional), <code>col</code> oder <code>colgroup</code> (optional), <code>thead</code> (optional), <code>tfoot</code> (optional), <code>tbody</code> (ein oder mehrere – wenn nur einmal benötigt, darf <code>tbody</code> auch entfallen, weshalb die herkömmliche Konstruktion, wonach <code>table</code> direkt aus <code>tr</code> -Elementen besteht, ebenfalls zulässig ist). |

**tbody**

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| Bedeutung      | Bestimmt in einer Tabelle den Tabellenkörper.                                               |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                    |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>table</code> .              |
| Kindelemente   | Darf nur folgendes andere HTML-Element (einfach oder mehrfach) enthalten: <code>tr</code> . |

***td***

|                |                                                                                    |
|----------------|------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Tabellenzelle.                                                       |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                           |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>tr</code> .        |
| Kindelemente   | Darf folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente]. |

***textarea***

|                |                                                                                                                                                                                           |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Definiert ein mehrzeiliges Eingabefeld.                                                                                                                                                   |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                  |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente] (außer <code>button</code> ), <code>body</code> ( <code>body</code> nur bei HTML transitional). |
| Kindelemente   | Darf nichts als <code>#PCDATA</code> enthalten.                                                                                                                                           |

***tfoot***

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| Bedeutung      | Bestimmt in einer Tabelle den Tabellenfuß.                                                  |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                    |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>table</code> .              |
| Kindelemente   | Darf nur folgendes andere HTML-Element (einfach oder mehrfach) enthalten: <code>tr</code> . |

***th***

|                |                                                                                    |
|----------------|------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Tabellenkopfzelle.                                                   |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                           |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>tr</code> .        |
| Kindelemente   | Darf folgende andere HTML-Elemente enthalten: [Block-Elemente], [Inline-Elemente]. |

***thead***

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| Bedeutung      | Bestimmt in einer Tabelle den Tabellenkopf.                                                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                    |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>table</code> .              |
| Kindelemente   | Darf nur folgendes andere HTML-Element (einfach oder mehrfach) enthalten: <code>tr</code> . |

***title***

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| Bedeutung      | Definiert den Titel einer HTML-Datei.                                         |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                      |
| Elternelemente | Darf nur innerhalb des folgenden HTML-Elements vorkommen: <code>head</code> . |
| Kindelemente   | Darf nichts als <code>#PCDATA</code> enthalten.                               |

**tr**

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| Bedeutung      | Definiert eine Tabellenzeile.                                                     |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: table, tbody, tfoot, thead. |
| Kindelemente   | Darf folgende anderen HTML-Elemente enthalten: td, th.                            |

**tt**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text, der im Schreibmaschinenstil dargestellt wird.                                                                |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

**u**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text als unterstrichen.                                                                                            |
| HTML-Varianten | Erlaubt in den Varianten transitional, frameset.                                                                                  |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

**ul**

|                |                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert eine Aufzählungsliste.                                                                                                                                                         |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                                                                                |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: applet, blockquote, body, button, center, dd, del, div, fieldset, form, iframe, ins, li, map, noframes, noscript, object, td, th. |
| Kindelemente   | Darf folgendes andere HTML-Element enthalten (ein- oder mehrmals): li.                                                                                                                  |

**var**

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Bedeutung      | Markiert einen Text als Variable.                                                                                                 |
| HTML-Varianten | Erlaubt in den Varianten strict, transitional, frameset.                                                                          |
| Elternelemente | Darf innerhalb der folgenden HTML-Elemente vorkommen: [Block-Elemente], [Inline-Elemente], body (body nur bei HTML transitional). |
| Kindelemente   | Darf #PCDATA und folgende andere HTML-Elemente enthalten: [Inline-Elemente].                                                      |

## C HTML: Attributreferenz

### C.1 Allgemeines zur HTML-Attributreferenz

Die meisten Attribute in HTML sind **implied** (optional), nur einige wenige sind **required** (im einleitenden HTML-Tag erforderlich). Bei den erforderlichen Attributen gibt es logisch notwendige (das `img`-Element würde beispielsweise ohne das `src`-Attribut wenig Sinn machen), aber auch »politische« (dass das W3-Konsortium etwa das `alt`-Attribut beim `img`-Element als notwendig einstuft, ist eindeutig ein erzieherischer Akt). Nicht wenige HTML-Attribute haben die Aufgabe, Elemente zu formatieren, z.B. Attribute wie `align=` (Ausrichtung) oder `bgcolor=` (Hintergrundfarbe). Solche Attribute sind jedoch als **deprecated** (*missbilligt*) eingestuft und nur noch in der HTML-Variante »transitional« erlaubt. Sie sollen allmählich aus dem HTML-Standard verschwinden, da solche Aufgaben durch CSS Stylesheets besser zu lösen sind.

### C.2 #CDATA

`#CDATA` steht für *character data*, zu deutsch: Zeichendaten. Es handelt sich um Text, der keine weiteren HTML-Elemente enthalten kann. In HTML ist das typischerweise Text, der bei einer Wertzuweisung an ein Attribut vorkommt. Solcher Text darf bei HTML 4.0 aus allen Zeichen bestehen, die in der Zeichenreferenz in Anhang D genannt sind. Benannte Zeichen wie `&uuml`; sind also ebenso erlaubt wie numerische Notationen von der Art `&#x20AC`; oder `&#8364`;. Solche Notationen werden erkannt und umgesetzt. HTML-Auszeichnungen, also die Notation von HTML-Elementen, wird in `CDATA`-Bereichen dagegen nicht erkannt. Das einzige, wonach gesucht wird und was erkannt wird, ist die Zeichenfolge `</`. Deshalb müssen Sie das Zeichen `<` in Wertzuweisungen an Attribute unbedingt wie üblich durch `&lt`; maskieren. Leerzeichen am Beginn oder Ende eines `#CDATA`-Textbereichs wird normalerweise abgeschnitten.

`#CDATA` schränkt zwar in keiner Weise ein, welcher Text einem Attribut als Wert zugewiesen wird. Dennoch sind in vielen Fällen nur bestimmte typische Angaben üblich. So fallen Sprachenkürzel beispielsweise auch unter `#CDATA`, aber erwünscht ist an der entsprechenden Stelle eine Angabe nach einem bestimmten Schema. Meistens werden solche Schemata durch RFCs geregelt. In dieser Attribut-Referenz wird auf solche RFCs hingewiesen.

### C.3 ID, IDREF oder Name

Der Wert eines solchen Attributs muss mit einem Buchstaben `A-Z` oder `a-z` beginnen und darf in der weiteren Zeichenkette ebenfalls nur Buchstaben `A-Z` und `a-z`, Ziffern `0-9`, Bindestriche (`-`), Unterstriche (`_`), Doppelpunkte (`:`) und Punkte (`.`) enthalten. Nicht erlaubt sind Sonderzeichen,

Leerzeichen oder andere Interpunktionszeichen. Werte vom Typ IDREF müssen als Wert den identisch geschriebenen Namen erhalten, der in einem anderen Element, auf das mit dem IDREF-Attribut Bezug genommen wird, mit dem Attribut `id=` vergeben wurde.

## C.4 Zahl

Als Wert eines solchen Attributs sind nur einfache ganzzahlige Werte erlaubt, die aus mindestens einer Ziffer 0-9 bestehen.

## C.5 Universalattribute

Universalattribute sind Attribute, die in der Mehrzahl aller HTML-Elemente vorkommen dürfen. Weitere Einzelheiten dazu bei den Beschreibungen zu Universalattributen in Kapitel 13.

### *Allgemeine Universalattribute*

| Attribut           | Wert                | erforderlich? | Bedeutung                                                                                                                                                                                                                                                                     |
|--------------------|---------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>class</code> | <code>#CDATA</code> | nein          | ordnet dem Element eine Stylesheet-Klasse zu, verwendbar in allen Elementen außer: <code>base</code> , <code>basefont</code> , <code>head</code> , <code>html</code> , <code>meta</code> , <code>param</code> , <code>script</code> , <code>style</code> , <code>title</code> |
| <code>id</code>    | <code>[ID]</code>   | nein          | ordnet dem Element einen individuellen Namen zu, verwendbar in allen Elementen außer: <code>base</code> , <code>head</code> , <code>html</code> , <code>meta</code> , <code>script</code> , <code>style</code> , <code>title</code>                                           |
| <code>style</code> | <code>#CDATA</code> | nein          | definiert CSS-Angaben zur Formatierung des Elements, verwendbar in allen Elementen außer: <code>base</code> , <code>basefont</code> , <code>head</code> , <code>html</code> , <code>meta</code> , <code>script</code> , <code>style</code> , <code>title</code>               |
| <code>title</code> | <code>#CDATA</code> | nein          | definiert einen Titel für das Element, verwendbar in allen Elementen außer: <code>base</code> , <code>basefont</code> , <code>head</code> , <code>html</code> , <code>meta</code> , <code>script</code> , <code>style</code> , <code>title</code>                             |

### *Universalattribute zur Internationalisierung*

| Attribut          | Wert                                | erforderlich? | Bedeutung                                                                                                                                                                                                                                                                                                                                   |
|-------------------|-------------------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dir</code>  | <code>ltr</code>   <code>rtl</code> | nein          | gibt die Schreibrichtung der im Element verwendeten Landessprache an, verwendbar in allen Elementen außer: <code>applet</code> , <code>base</code> , <code>basefont</code> , <code>br</code> , <code>frame</code> , <code>frameset</code> , <code>hr</code> , <code>iframe</code> , <code>param</code> , <code>script</code>                |
| <code>lang</code> | <code>#CDATA</code>                 | nein          | gibt die im Element verwendete Landessprache (nach RFC 1766) an, verwendbar in allen Elementen außer: <code>applet</code> , <code>base</code> , <code>basefont</code> , <code>br</code> , <code>frame</code> , <code>frameset</code> , <code>hr</code> , <code>iframe</code> , <code>meta</code> , <code>param</code> , <code>script</code> |



**Universalattribute für Event-Handler**

| Attribut    | Wert   | erforderlich? | Bedeutung                                                                                                                                                                                                       |
|-------------|--------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| onclick     | #CDATA | nein          | beim Anklicken des Elements,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title                   |
| ondblclick  | #CDATA | nein          | beim doppelten Anklicken des Elements,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title         |
| onmousedown | #CDATA | nein          | bei gedrückter Maustaste über dem Element,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title     |
| onmouseup   | #CDATA | nein          | bei Loslassen der Maustaste über dem Element,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title  |
| onmouseover | #CDATA | nein          | beim Überfahren des Elements mit der Maus,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title     |
| onmousemove | #CDATA | nein          | beim Ziehen des Elements mit der Maus,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title         |
| onmouseout  | #CDATA | nein          | beim Verlassen des Elements mit der Maus,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title      |
| onkeypress  | #CDATA | nein          | beim Drücken und wieder Loslassen einer Taste,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title |

| Attribut  | Wert   | erforderlich? | Bedeutung                                                                                                                                                                                    |
|-----------|--------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| onkeydown | #CDATA | nein          | beim Drücken einer Taste,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title   |
| onkeyup   | #CDATA | nein          | beim Loslassen einer Taste,<br>verwendbar in allen Elementen außer: applet,<br>base, basefont, bdo, br, font, frame,<br>frameset, head, html, iframe, isindex,<br>param, script style, title |

## C.6 Attributreferenz

### a

| Bedeutung | Definiert einen Anker oder einen Hyperlink |                                   |                          |                                                                                |
|-----------|--------------------------------------------|-----------------------------------|--------------------------|--------------------------------------------------------------------------------|
| Attribut  | Wert                                       | HTML-Varianten                    | erforderlich?            | Bedeutung                                                                      |
| accesskey | #CDATA                                     | strict, transitional,<br>frameset | nein                     | bestimmt ein einzelnes Zeichen als<br>Hotkey für den Hyperlink                 |
| charset   | #CDATA                                     | strict, transitional,<br>frameset | nein                     | gibt den Zeichensatz des Verweis-<br>ziels (laut RFC 2045) an                  |
| coords    | #CDATA                                     | strict, transitional,<br>frameset | nein                     | verweissensitive Bereiche bei Ob-<br>jekten                                    |
| href      | #CDATA                                     | strict, transitional,<br>frameset | ja (bei Hyper-<br>links) | gibt das Verweisziel (nach RFC<br>2396) an                                     |
| hreflang  | #CDATA                                     | strict, transitional,<br>frameset | nein                     | gibt die Landessprache des Ver-<br>weisziels (nach RFC 1766) an                |
| name      | #CDATA                                     | strict, transitional,<br>frameset | ja (bei An-<br>kern)     | bestimmt den Ankernamen                                                        |
| rel       | #CDATA                                     | strict, transitional,<br>frameset | nein                     | bestimmt das Verweisziel als<br>»Vorwärtsverknüpfung«                          |
| rev       | #CDATA                                     | strict, transitional,<br>frameset | nein                     | bestimmt das Verweisziel als<br>»Rückwärtsverknüpfung«                         |
| shape     | rect<br>circle<br>poly<br>default          | strict, transitional,<br>frameset | nein (Default:<br>rect)  | verweissensitive Bereiche bei Ob-<br>jekten                                    |
| tabindex  | [Zahl]                                     | strict, transitional,<br>frameset | nein                     | bestimmt die Tabulatorreihenfolge<br>für Hyperlinks durch Angabe einer<br>Zahl |

| <i>Bedeutung</i>     | <i>Definiert einen Anker oder einen Hyperlink</i> |                                   |      |                                                         |
|----------------------|---------------------------------------------------|-----------------------------------|------|---------------------------------------------------------|
| target               | #CDATA<br>_blank<br>_parent<br>_self<br>_top      | strict, transitional,<br>frameset | nein | bestimmt den Fensternamen des Verweisziels              |
| type                 | #CDATA                                            | strict, transitional,<br>frameset | nein | bestimmt den Mime-Type des Verweisziels (laut RFC 2045) |
| [Universalattribute] |                                                   |                                   |      |                                                         |

**abbr**

| <i>Bedeutung</i>     | <i>Markiert eine Abkürzung</i> |
|----------------------|--------------------------------|
| [Universalattribute] |                                |

**acronym**

| <i>Bedeutung</i>     | <i>Markiert eine Abkürzung</i> |
|----------------------|--------------------------------|
| [Universalattribute] |                                |

**address**

| <i>Bedeutung</i>     | <i>Markiert eine Abkürzung</i> |
|----------------------|--------------------------------|
| [Universalattribute] |                                |

**applet**

| <i>Bedeutung</i> | <i>Fügt ein Java-Applet ein</i>                    |                        |                      |                                                                     |
|------------------|----------------------------------------------------|------------------------|----------------------|---------------------------------------------------------------------|
| <i>Attribut</i>  | <i>Wert</i>                                        | <i>HTML-Varianten</i>  | <i>erforderlich?</i> | <i>Bedeutung</i>                                                    |
| align            | top<br>middle<br>bottom<br>left<br>center<br>right | transitional, frameset | nein                 | richtet das Applet aus                                              |
| alt              | #CDATA                                             | transitional, frameset | nein                 | bestimmt einen Alternativtext, falls das Applet nicht anzeigbar ist |

## 1230 Anhang C: HTML: Attributreferenz

| Bedeutung            | Fügt ein Java-Applet ein |                        |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|--------------------------|------------------------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| archive              | #CDATA                   | transitional, frameset | nein | gibt eine kommaseparierte Liste mit Archivdateien für das Applet an<br>gibt die Klassendatei des Applets an<br>gibt die Anzeigehöhe des Applets in Pixeln oder Prozent an<br>bestimmt den horizontalen Abstand zwischen Applet und umfließendem Text in Pixeln oder Prozent<br>gibt einen Namen für das Applet an<br>bestimmt den horizontalen Abstand zwischen Applet und umfließendem Text in Pixeln oder Prozent<br>gibt die Anzeigebreite des Applets in Pixeln oder Prozent an |
| code                 | #CDATA                   | transitional, frameset | nein |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| height               | #CDATA                   | transitional, frameset | ja   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| hspace               | #CDATA                   | transitional, frameset | nein |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| name                 | #CDATA                   | transitional, frameset | nein |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| vspace               | #CDATA                   | transitional, frameset | nein |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| width                | #CDATA                   | transitional, frameset | ja   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| [Universalattribute] |                          |                        |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

### area

| <i>Bedeutung</i> | <i>Definiert einen Ausschnitt für eine verweissensitive Grafik</i> |                                |                      |                                                                                          |
|------------------|--------------------------------------------------------------------|--------------------------------|----------------------|------------------------------------------------------------------------------------------|
| <i>Attribut</i>  | <i>Wert</i>                                                        | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                         |
| alt              | #CDATA                                                             | strict, transitional, frameset | ja                   | Kurzbeschreibung des Verzeisziels                                                        |
| accesskey        | #CDATA                                                             | strict, transitional, frameset | nein                 | bestimmt ein einzelnes Zeichen als Hotkey für den Hyperlink                              |
| coords           | #CDATA                                                             | strict, transitional, frameset | nein                 | bestimmt die Koordinaten des verweissensitiven Bereichs, abhängig von <code>shape</code> |
| href             | #CDATA                                                             | strict, transitional, frameset | nein                 | gibt das Verweisziel (nach RFC 2396) an                                                  |
| nohref           | (nohref)                                                           | strict, transitional, frameset | nein                 | bestimmt für einen verweissensitiven Bereich, dass keine Aktion damit verbunden ist      |
| onblur           | #CDATA                                                             | strict, transitional, frameset | nein                 | beim Verlassen des verweissensitiven Bereichs... (Script-Code)                           |
| onfocus          | #CDATA                                                             | strict, transitional, frameset | nein                 | wenn verweissensitiver Bereich den Fokus hat... (Script-Code)                            |
| shape            | rect<br>circle<br>poly<br>default                                  | strict, transitional, frameset | nein (Default: rect) | bestimmt den Typ eines verweissensitiven Bereichs                                        |
| tabindex         | [Zahl]                                                             | strict, transitional, frameset | nein                 | bestimmt die Tabulatorreihenfolge für Hyperlinks                                         |

**Bedeutung** *Definiert einen Ausschnitt für eine verweissensitive Grafik*

|        |         |                                   |      |                                               |
|--------|---------|-----------------------------------|------|-----------------------------------------------|
| target | #CDATA  | strict, transitional,<br>frameset | nein | bestimmt den Fensternamen des<br>Verweisziels |
|        | _blank  |                                   |      |                                               |
|        | _parent |                                   |      |                                               |
|        | _self   |                                   |      |                                               |
|        | _top    |                                   |      |                                               |

[Universalattribute]

**b****Bedeutung** *Markiert fett gedruckten Text*

[Universalattribute]

**base****Bedeutung** *Definiert einen Ausschnitt für eine verweissensitive Grafik*

| Attribut | Wert   | HTML-Varianten                    | erforderlich? | Bedeutung                                       |
|----------|--------|-----------------------------------|---------------|-------------------------------------------------|
| href     | #CDATA | strict, transitional,<br>frameset | nein          | gibt den Basis-URI an                           |
| target   | #CDATA | strict, transitional,<br>frameset | nein          | gibt das Default-Zielfenster für<br>Verweise an |

**basefont****Bedeutung** *Bestimmt Basis-Schriftmerkmale für nachfolgenden Text*

| Attribut | Wert   | HTML-Varianten            | erforderlich? | Bedeutung                                                                                   |
|----------|--------|---------------------------|---------------|---------------------------------------------------------------------------------------------|
| color    | #CDATA | transitional,<br>frameset | nein          | gibt die Basisschriftfarbe hexadezi-<br>mal oder als Farbwort an                            |
| face     | #CDATA | transitional,<br>frameset | nein          | gibt die Basisschriftart an                                                                 |
| size     | #CDATA | transitional,<br>frameset | ja            | gibt die Basisschriftgröße als Zahl als<br>Zahl absolut oder relativ mit Vor-<br>zeichen an |

[Universalattribute]

## 1232 Anhang C: HTML: Attributreferenz

### **bdo**

| Bedeutung      Markiert eine Änderung der Textrichtung bei Sprachen mit anderer Schreibrichtung |            |                                      |               |                             |
|-------------------------------------------------------------------------------------------------|------------|--------------------------------------|---------------|-----------------------------|
| Attribut                                                                                        | Wert       | HTML-Varianten                       | erforderlich? | Bedeutung                   |
| dir                                                                                             | ltr<br>rtl | strict,<br>transitional,<br>frameset | ja            | gibt die Schreibrichtung an |
| Universalattribute: id class style title                                                        |            |                                      |               |                             |

### **big**

| Bedeutung            | Markiert Text, der größer dargestellt werden soll als normal |
|----------------------|--------------------------------------------------------------|
| [Universalattribute] |                                                              |

### **blockquote**

| Bedeutung      Markiert ein Zitat |        |                                      |               |                                                 |
|-----------------------------------|--------|--------------------------------------|---------------|-------------------------------------------------|
| Attribut                          | Wert   | HTML-Varianten                       | erforderlich? | Bedeutung                                       |
| cite                              | #CDATA | strict,<br>transitional,<br>frameset | nein          | gibt den URI der Zitatquelle (nach RFC 2396) an |
| [Universalattribute]              |        |                                      |               |                                                 |

### **body**

| Bedeutung      Markiert den Dokumentkörper und damit den Bereich der darzustellenden Daten |        |                           |               |                                                                                                          |
|--------------------------------------------------------------------------------------------|--------|---------------------------|---------------|----------------------------------------------------------------------------------------------------------|
| Attribut                                                                                   | Wert   | HTML-Varianten            | erforderlich? | Bedeutung                                                                                                |
| alink                                                                                      | #CDATA | transitional,<br>frameset | nein          | gibt die dokumentweite Farbe für aktivierte Links hexadezimal (in der Form #XXXXXX) oder als Farbwert an |
| background                                                                                 | #CDATA | transitional,<br>frameset | nein          | gibt den URI einer dokumentweiten Hintergrundgrafik (nach RFC 2396) an                                   |
| bgcolor                                                                                    | #CDATA | transitional,<br>frameset | nein          | gibt die dokumentweite Hintergrundfarbe hexadezimal (in der Form #XXXXXX) oder als Farbwert an           |

| <i>Bedeutung</i> <i>Markiert den Dokumentkörper und damit den Bereich der darzustellenden Daten</i> |        |                                      |      |                                                                                                                              |
|-----------------------------------------------------------------------------------------------------|--------|--------------------------------------|------|------------------------------------------------------------------------------------------------------------------------------|
| link                                                                                                | #CDATA | transitional,<br>frameset            | nein | gibt die dokumentweite Farbe für Links zu noch nicht besuchten Seiten hexadezimal (in der Form #XXXXXX) oder als Farbwert an |
| onload                                                                                              | #CDATA | strict,<br>transitional,<br>frameset | nein | wenn die Datei geladen ist... (Script-Code)                                                                                  |
| onunload                                                                                            | #CDATA | strict,<br>transitional,<br>frameset | nein | wenn die Datei verlassen wird... (Script-Code)                                                                               |
| text                                                                                                | #CDATA | transitional,<br>frameset            | nein | gibt die dokumentweite Textfarbe hexadezimal (in der Form #XXXXXX) oder als Farbwert an                                      |
| vlink                                                                                               | #CDATA | transitional,<br>frameset            | nein | gibt die dokumentweite Farbe für Links zu bereits besuchten Seiten hexadezimal (in der Form #XXXXXX) oder als Farbwert an    |
| [Universalattribute]                                                                                |        |                                      |      |                                                                                                                              |

**br**

| <i>Bedeutung</i> <i>Erzeugt einen Zeilenumbruch</i> |                              |                           |                            |                                                                                        |
|-----------------------------------------------------|------------------------------|---------------------------|----------------------------|----------------------------------------------------------------------------------------|
| <i>Attribut</i>                                     | <i>Wert</i>                  | <i>HTML-Varianten</i>     | <i>erforderlich?</i>       | <i>Bedeutung</i>                                                                       |
| clear                                               | left<br>all<br>right<br>none | transitional,<br>frameset | nein<br>(Default:<br>none) | bestimmt bei Text, der um eine Grafik oder ein Objekt fließt, die Fortsetzungsposition |
| [Universalattribute]                                |                              |                           |                            |                                                                                        |

**button**

| <i>Bedeutung</i> <i>Erzeugt einen Klick-Button</i> |             |                                   |                      |                                                                                                       |
|----------------------------------------------------|-------------|-----------------------------------|----------------------|-------------------------------------------------------------------------------------------------------|
| <i>Attribut</i>                                    | <i>Wert</i> | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                                      |
| accesskey                                          | #CDATA      | strict, transitional,<br>frameset | nein                 | bestimmt ein einzelnes Zeichen als Hotkey für den Tastaturzugriff auf den Button                      |
| datafld                                            | #CDATA      | strict, transitional,<br>frameset | nein                 | gibt den Namen eines Datenfeldes des aktuellen Datensatzes der mit datasrc= bestimmten Datenquelle an |

## 1234 Anhang C: HTML: Attributreferenz

| <i>Bedeutung Erzeugt einen Klick-Button</i> |                                   |                                       |                                  |                                                                                                                                      |
|---------------------------------------------|-----------------------------------|---------------------------------------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>datasrc</i>                              | <i>#CDATA</i>                     | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>gibt den URI einer Datenquelle (nach RFC 2396) an, die mit dem Button verknüpft ist</i>                                           |
| <i>dataformatas</i>                         | <i>plaintext   html</i>           | <i>strict, transitional, frameset</i> | <i>nein (Default: plaintext)</i> | <i>gibt an, wie die Daten, die aufgrund der Angaben <i>datasrc=</i> und <i>datafld=</i> ermittelt werden, zu interpretieren sind</i> |
| <i>disabled</i>                             | <i>disabled (leeres Attribut)</i> | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>bestimmt, dass der Button nicht betätigt werden kann</i>                                                                          |
| <i>name</i>                                 | <i>#CDATA</i>                     | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>bestimmt einen Namen für den Button</i>                                                                                           |
| <i>onblur</i>                               | <i>#CDATA</i>                     | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>beim Verlassen des verweissensitiven Bereichs... (Script-Code)</i>                                                                |
| <i>onfocus</i>                              | <i>#CDATA</i>                     | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>wenn verweissensitiver Bereich den Fokus hat... (Script-Code)</i>                                                                 |
| <i>tabindex</i>                             | <i>[Zahl]</i>                     | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>bestimmt die Tabulatorreihenfolge für Buttons durch Angabe einer Zahl</i>                                                         |
| <i>type</i>                                 | <i>button   submit   reset</i>    | <i>strict, transitional, frameset</i> | <i>nein (Default: submit)</i>    | <i>bestimmt den Typ des Buttons</i>                                                                                                  |
| <i>value</i>                                | <i>#CDATA</i>                     | <i>strict, transitional, frameset</i> | <i>nein</i>                      | <i>bestimmt den Absendewert des Buttons</i>                                                                                          |
| [Universalattribute]                        |                                   |                                       |                                  |                                                                                                                                      |

### **caption**

| <i>Bedeutung Setzt eine Tabellenüberschrift/Tabellenunterschrift</i> |                                          |                               |                      |                                            |
|----------------------------------------------------------------------|------------------------------------------|-------------------------------|----------------------|--------------------------------------------|
| <i>Attribut</i>                                                      | <i>Wert</i>                              | <i>HTML-Varianten</i>         | <i>erforderlich?</i> | <i>Bedeutung</i>                           |
| <i>align</i>                                                         | <i>top<br/>bottom<br/>left<br/>right</i> | <i>transitional, frameset</i> | <i>nein</i>          | <i>richtet die Tabellenüberschrift aus</i> |
| [Universalattribute]                                                 |                                          |                               |                      |                                            |

### **center**

| <i>Bedeutung Markiert einen Bereich, der zentriert dargestellt wird</i> |  |
|-------------------------------------------------------------------------|--|
| [Universalattribute]                                                    |  |



***cite***

|                  |                           |
|------------------|---------------------------|
| <b>Bedeutung</b> | <i>Markiert ein Zitat</i> |
|------------------|---------------------------|

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

***code***

|                  |                                                          |
|------------------|----------------------------------------------------------|
| <b>Bedeutung</b> | <i>Markiert Text als Quelltext einer Computersprache</i> |
|------------------|----------------------------------------------------------|

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

***col***

|                  |                                                                     |  |  |  |
|------------------|---------------------------------------------------------------------|--|--|--|
| <b>Bedeutung</b> | <i>Bestimmt Eigenschaften für eine oder mehrere Tabellenspalten</i> |  |  |  |
|------------------|---------------------------------------------------------------------|--|--|--|

| <i>Attribut</i> | <i>Wert</i>                                | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                              |
|-----------------|--------------------------------------------|-----------------------------------|----------------------|-------------------------------------------------------------------------------|
| align           | left<br>center<br>right<br>justify<br>char | strict, transitional,<br>frameset | nein                 | richtet Tabellenspalten horizontal aus                                        |
| char            | #CDATA                                     | strict, transitional,<br>frameset | nein                 | bestimmt in Verbindung mit align="char" das Ausrichtungszeichen               |
| charoff         | #CDATA                                     | strict, transitional,<br>frameset | nein                 | bestimmt in Verbindung mit align="char" die Position des Ausrichtungszeichens |
| span            | #CDATA                                     | strict, transitional,<br>frameset | nein                 | gibt an, für wie viele Spalten die Angaben gelten                             |
| valign          | top<br>middle<br>bottom<br>baseline        | strict, transitional,<br>frameset | nein                 | richtet Tabellenspalten vertikal aus                                          |
| width           | #CDATA                                     | strict, transitional,<br>frameset | nein                 | bestimmt die Spaltenbreite                                                    |

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**colgroup**

| <i>Bedeutung</i> <i>Bestimmt Default-Eigenschaften für eine Gruppe von Tabellenspalten</i> |                                            |                                   |                      |                                                                                              |
|--------------------------------------------------------------------------------------------|--------------------------------------------|-----------------------------------|----------------------|----------------------------------------------------------------------------------------------|
| <i>Attribut</i>                                                                            | <i>Wert</i>                                | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                             |
| align                                                                                      | left<br>center<br>right<br>justify<br>char | strict, transitional,<br>frameset | nein                 | gibt einen Wert für die horizontale Ausrichtung der Tabellenspalten vor                      |
| char                                                                                       | #CDATA                                     | strict, transitional,<br>frameset | nein                 | gibt einen Wert für das Ausrichtungszeichen in Verbindung mit align="char" vor               |
| charoff                                                                                    | #CDATA                                     | strict, transitional,<br>frameset | nein                 | gibt einen Wert für die Position des Ausrichtungszeichens in Verbindung mit align="char" vor |
| span                                                                                       | [Zahl]                                     | strict, transitional,<br>frameset | nein                 | gibt an, für wie viele Spalten die Angaben per Default gelten                                |
| valign                                                                                     | top<br>middle<br>bottom<br>baseline        | strict, transitional,<br>frameset | nein                 | gibt einen Wert für die vertikale Ausrichtung der Tabellenspalten vor                        |
| width                                                                                      | #CDATA                                     | strict, transitional,<br>frameset | nein                 | gibt einen Wert für die Breite der Tabellenspalten vor                                       |
| [Universalattribute]                                                                       |                                            |                                   |                      |                                                                                              |

**dd**

| <i>Bedeutung</i>     | <i>Markiert in einer Definitionsliste eine Definition zu einem Term</i> |
|----------------------|-------------------------------------------------------------------------|
| [Universalattribute] |                                                                         |

**del**

| <i>Bedeutung</i> <i>Markiert Text als gelöscht im Sinne von Änderungsmarkierung</i> |             |                                   |                      |                                                                            |
|-------------------------------------------------------------------------------------|-------------|-----------------------------------|----------------------|----------------------------------------------------------------------------|
| <i>Attribut</i>                                                                     | <i>Wert</i> | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                           |
| cite                                                                                | #CDATA      | strict, transitional,<br>frameset | nein                 | gibt den URI (nach RFC 2396) an, wo Gründe für die Änderung angegeben sind |
| datetime                                                                            | #CDATA      | strict, transitional,<br>frameset | nein                 | gibt Datum und Uhrzeit der Änderung nach ISO-Format an                     |
| [Universalattribute]                                                                |             |                                   |                      |                                                                            |

**dfn**

| Bedeutung | Markiert eine Definition |
|-----------|--------------------------|
|-----------|--------------------------|

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**dir**

| Bedeutung | Markiert eine Verzeichnisliste |
|-----------|--------------------------------|
|-----------|--------------------------------|

| Attribut | Wert                         | HTML-Varianten         | erforderlich? | Bedeutung                                                    |
|----------|------------------------------|------------------------|---------------|--------------------------------------------------------------|
| compact  | compact<br>(leeres Attribut) | transitional, frameset | nein          | bestimmt, dass die Liste in engerer Schrift dargestellt wird |

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**div**

| Bedeutung | Markiert einen allgemeinen Bereich |
|-----------|------------------------------------|
|-----------|------------------------------------|

| Attribut     | Wert                               | HTML-Varianten                 | erforderlich?                | Bedeutung                                                                                                                                 |
|--------------|------------------------------------|--------------------------------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| align        | left<br>center<br>right<br>justify | transitional, frameset         | nein<br>(Default: left)      | richtet den Inhalt des Bereichs aus                                                                                                       |
| datafld      | #CDATA                             | strict, transitional, frameset | nein                         | gibt den Namen eines Datenfeldes des aktuellen Datensatzes der mit <code>data-source</code> bestimmten Datenquelle an                     |
| datasrc      | #CDATA                             | strict, transitional, frameset | nein                         | gibt den URI einer Datenquelle (nach RFC 2396) an, die mit dem Bereich verknüpft ist                                                      |
| dataformatas | plain-text<br>html                 | strict, transitional, frameset | nein<br>(Default: plaintext) | gibt an, wie die Daten, die aufgrund der Angaben <code>datasrc=</code> und <code>datafld=</code> ermittelt werden, zu interpretieren sind |

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**dl**

| Bedeutung | Markiert eine Definitionsliste |
|-----------|--------------------------------|
|-----------|--------------------------------|

| Attribut | Wert                         | HTML-Varianten         | erforderlich? | Bedeutung                                                    |
|----------|------------------------------|------------------------|---------------|--------------------------------------------------------------|
| compact  | compact<br>(leeres Attribut) | transitional, frameset | nein          | bestimmt, dass die Liste in engerer Schrift dargestellt wird |

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**dt**

**Bedeutung**     *Markiert in einer Definitionsliste einen zu definierenden Term*

[Universalattribute]

**em**

**Bedeutung**     *Markiert Text als emphatisch*

[Universalattribute]

**fieldset**

**Bedeutung**     *Markiert eine Gruppe von Formularelementen*

[Universalattribute]

**font**

**Bedeutung**     *Markiert Text mit Schriftgröße, Schriftfarbe und Schriftart*

| Attribut | Wert   | HTML-Varianten            | erforderlich? | Bedeutung                                                                         |
|----------|--------|---------------------------|---------------|-----------------------------------------------------------------------------------|
| color    | #CDATA | transitional,<br>frameset | nein          | gibt die Schriftfarbe hexadezimal<br>(#XXXXXX) oder als Farbwert an               |
| face     | #CDATA | transitional,<br>frameset | nein          | gibt die Schriftart an                                                            |
| size     | #CDATA | transitional,<br>frameset | ja            | gibt die Schriftgröße als Zahl als Zahl<br>absolut oder relativ mit Vorzeichen an |

[Universalattribute]

**form**

**Bedeutung**     *Definiert ein Formular*

| Attribut       | Wert   | HTML-Varianten                    | erforderlich? | Bedeutung                                                                   |
|----------------|--------|-----------------------------------|---------------|-----------------------------------------------------------------------------|
| action         | #CDATA | strict, transitional,<br>frameset | ja            | gibt einen URI (nach RFC 2396) für die<br>Verarbeitung der Formulardaten an |
| accept         | #CDATA | strict, transitional,<br>frameset | nein          | listet erlaubte Mime-Typen für File-<br>Upload (nach RFC 2045) auf          |
| accept-charset | #CDATA | strict, transitional,<br>frameset | nein          | listet unterstützte Zeichensätze (nach<br>RFC 2045) auf                     |

| Bedeutung            | Definiert ein Formular                       |                                |                                                      |                                                                             |
|----------------------|----------------------------------------------|--------------------------------|------------------------------------------------------|-----------------------------------------------------------------------------|
| enctype              | #CDATA                                       | strict, transitional, frameset | nein<br>(Default: application/x-www-form-urlencoded) | gibt den Mime-Type (nach RFC 2045) an, mit dem das Formular übertragen wird |
| method               | get<br>post                                  | strict, transitional, frameset | nein<br>(Default: get)                               | bestimmt, wie Formulardaten auf dem Server zur Verfügung stehen             |
| name                 | #CDATA                                       | strict, transitional, frameset | nein                                                 | bestimmt einen Namen für das Formular                                       |
| onreset              | #CDATA                                       | strict, transitional, frameset | nein                                                 | beim Zurücksetzen des Formulars (Script-Code)                               |
| onsubmit             | #CDATA                                       | strict, transitional, frameset | nein                                                 | beim Absenden des Formulars (Script-Code)                                   |
| target               | #CDATA<br>_blank<br>_parent<br>_self<br>_top | strict, transitional, frameset | nein                                                 | bestimmt den Fensternamen für Antwortsauagen                                |
| [Universalattribute] |                                              |                                |                                                      |                                                                             |

## frame

| Bedeutung    | Definiert ein Frame-Fenster   |                        |                   |                                                                                                             |
|--------------|-------------------------------|------------------------|-------------------|-------------------------------------------------------------------------------------------------------------|
| Attribut     | Wert                          | HTML-Varianten         | erforderlich?     | Bedeutung                                                                                                   |
| frameborder  | 1<br>0                        | transitional, frameset | nein (Default: 1) | gibt an, ob das Frame-Fenster einen sichtbaren Rahmen haben soll oder nicht                                 |
| longdesc     | #CDATA                        | transitional, frameset | nein              | gibt einen URI (nach RFC 2396) an, wo eine längere Beschreibung zum Inhalt des Frame-Fensters zu finden ist |
| marginwidth  | #CDATA                        | transitional, frameset | nein              | gibt die Anzahl Pixel für den Abstand des Fensterinhalts zum linken und rechten Fensterrand an              |
| marginheight | #CDATA                        | transitional, frameset | nein              | gibt die Anzahl Pixel für den Abstand des Fensterinhalts zum oberen und unteren Fensterrand an              |
| name         | #CDATA                        | transitional, frameset | nein              | bestimmt einen Namen für das Frame-Fenster                                                                  |
| noresize     | noresize<br>(leeres Attribut) | transitional, frameset | nein              | wenn angegeben, kann der Anwender die Größe des Frame-Fensters nicht verändern                              |

## 1240 Anhang C: HTML: Attributreferenz

| Bedeutung            |        | Definiert ein Frame-Fenster |                         |                                                                             |
|----------------------|--------|-----------------------------|-------------------------|-----------------------------------------------------------------------------|
| scrolling            | yes    | transitional,<br>frameset   | nein (Default:<br>auto) | bestimmt, ob das Frame-Fenster Scrollbars erhält                            |
|                      | no     |                             |                         |                                                                             |
|                      | auto   |                             |                         |                                                                             |
| src                  | #CDATA | transitional,<br>frameset   | nein                    | bestimmt einen URI (nach RFC 2396), die in dem Frame-Fenster angezeigt wird |
| [Universalattribute] |        |                             |                         |                                                                             |

### frameset

| Bedeutung            |        | Definiert ein Frame-Fenster |                         |                                                                                              |
|----------------------|--------|-----------------------------|-------------------------|----------------------------------------------------------------------------------------------|
| Attribut             | Wert   | HTML-Varianten              | erforderlich?           | Bedeutung                                                                                    |
| cols                 | #CDATA | transitional,<br>frameset   | nein (Default:<br>100%) | bestimmt für ein oder mehrere spaltenartig angeordnete Frame-Fenster die Breitenverhältnisse |
| onload               | #CDATA | transitional,<br>frameset   | nein                    | wenn das Frameset geladen ist...<br>(Script-Code)                                            |
| onunload             | #CDATA | transitional,<br>frameset   | nein                    | wenn das Frameset verlassen wird...<br>(Script-Code)                                         |
| rows                 | #CDATA | transitional,<br>frameset   | nein                    | bestimmt für ein oder mehrere reihenartig angeordnete Frame-Fenster die Höhenverhältnisse    |
| [Universalattribute] |        |                             |                         |                                                                                              |

### h1-h6

| Bedeutung            | Markiert eine Überschrift 1. bis 6. Ordnung |                           |               |                                        |
|----------------------|---------------------------------------------|---------------------------|---------------|----------------------------------------|
| Attribut             | Wert                                        | HTML-Varianten            | erforderlich? | Bedeutung                              |
| align                | left                                        | transitional,<br>frameset | nein          | richtet den Inhalt der Überschrift aus |
|                      | center                                      |                           | (Default:     |                                        |
|                      | right                                       |                           | left)         |                                        |
|                      | justify                                     |                           |               |                                        |
| [Universalattribute] |                                             |                           |               |                                        |

### head

| Bedeutung                    |        | Markiert den Kopfbereich einer HTML-Datei |               |                                                                        |
|------------------------------|--------|-------------------------------------------|---------------|------------------------------------------------------------------------|
| Attribut                     | Wert   | HTML-Varianten                            | erforderlich? | Bedeutung                                                              |
| profile                      | #CDATA | strict, transitional,<br>frameset         | nein          | gibt ein Verzeichnis für Meta-Informationen als URI (nach RFC 2396) an |
| Universalattribute: dir lang |        |                                           |               |                                                                        |

**hr**

| <i>Bedeutung</i> <i>Erzeugt eine horizontale Trennlinie</i> |                                 |                           |                      |                                                                             |
|-------------------------------------------------------------|---------------------------------|---------------------------|----------------------|-----------------------------------------------------------------------------|
| <i>Attribut</i>                                             | <i>Wert</i>                     | <i>HTML-Varianten</i>     | <i>erforderlich?</i> | <i>Bedeutung</i>                                                            |
| align                                                       | left                            | transitional,             | nein                 | richtet die Trennlinie aus                                                  |
|                                                             | center                          | frameset                  | (Default:            |                                                                             |
|                                                             | right                           |                           | left)                |                                                                             |
| noshade                                                     | noshade<br>(leeres<br>Attribut) | transitional,<br>frameset | nein                 | bestimmt, dass die Trennlinie flach<br>und ohne 3-D-Effekt dargestellt wird |
| size                                                        | #CDATA                          | transitional,<br>frameset | nein                 | bestimmt die Dicke der Trennlinie in<br>Pixeln                              |
| width                                                       | #CDATA                          | transitional,<br>frameset | nein                 | bestimmt die Länge der Trennlinie in<br>Pixeln oder Prozent                 |
| [Universalattribute]                                        |                                 |                           |                      |                                                                             |

**html**

| <i>Bedeutung</i> <i>Wurzelement einer HTML-Datei</i> |             |                           |                      |                                     |
|------------------------------------------------------|-------------|---------------------------|----------------------|-------------------------------------|
| <i>Attribut</i>                                      | <i>Wert</i> | <i>HTML-Varianten</i>     | <i>erforderlich?</i> | <i>Bedeutung</i>                    |
| version                                              | #CDATA      | transitional,<br>frameset | nein                 | gibt die verwendete HTML-Version an |
| Universalattribute: dir   lang                       |             |                           |                      |                                     |

**i**

| <i>Bedeutung</i> <i>Markiert kursiv gedruckten Text</i> |  |
|---------------------------------------------------------|--|
| [Universalattribute]                                    |  |

**iframe**

| <i>Bedeutung</i> <i>Definiert ein eingebettetes Fenster</i> |             |                       |                      |                        |
|-------------------------------------------------------------|-------------|-----------------------|----------------------|------------------------|
| <i>Attribut</i>                                             | <i>Wert</i> | <i>HTML-Varianten</i> | <i>erforderlich?</i> | <i>Bedeutung</i>       |
| align                                                       | top         | transitional,         | nein                 | richtet die Grafik aus |
|                                                             | middle      | frameset              |                      |                        |
|                                                             | bottom      |                       |                      |                        |
|                                                             | left        |                       |                      |                        |
|                                                             | center      |                       |                      |                        |
|                                                             | right       |                       |                      |                        |
|                                                             |             |                       |                      |                        |

| <i>Bedeutung</i> | <i>Definiert ein eingebettetes Fenster</i> |                           |                         |                                                                                                             |
|------------------|--------------------------------------------|---------------------------|-------------------------|-------------------------------------------------------------------------------------------------------------|
| frameborder      | 1<br>0                                     | transitional,<br>frameset | nein<br>(Default: 1)    | gibt an, ob das Frame-Fenster einen sichtbaren Rahmen haben soll oder nicht                                 |
| height           | #CDATA                                     | transitional,<br>frameset | nein                    | bestimmt die Höhe des Frame-Fensters in Pixeln oder Prozent                                                 |
| longdesc         | #CDATA                                     | transitional,<br>frameset | nein                    | gibt einen URI (nach RFC 2396) an, wo eine längere Beschreibung zum Inhalt des Frame-Fensters zu finden ist |
| marginwidth      | #CDATA                                     | transitional,<br>frameset | nein                    | gibt die Anzahl Pixel für den Abstand des Fensterinhalts zum linken und rechten Fensterrand an              |
| marginheight     | #CDATA                                     | transitional,<br>frameset | nein                    | gibt die Anzahl Pixel für den Abstand des Fensterinhalts zum oberen und unteren Fensterrand an              |
| name             | #CDATA                                     | transitional,<br>frameset | nein                    | bestimmt einen Namen für das Frame-Fenster                                                                  |
| scrolling        | yes<br>no<br>auto                          | transitional,<br>frameset | nein<br>(Default: auto) | bestimmt, ob das Frame-Fenster Scrollbars erhält                                                            |
| src              | #CDATA                                     | transitional,<br>frameset | nein                    | bestimmt einen URI (nach RFC 2396), die in dem Frame-Fenster angezeigt wird                                 |
| width            | #CDATA                                     | transitional,<br>frameset | nein                    | bestimmt die Breite des Frame-Fensters in Pixeln oder Prozent                                               |

[Universalattribute]

**img**

| <i>Bedeutung</i> | <i>Referenziert eine Grafik</i>                    |                                   |                      |                                                                     |
|------------------|----------------------------------------------------|-----------------------------------|----------------------|---------------------------------------------------------------------|
| <i>Attribut</i>  | <i>Wert</i>                                        | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                    |
| align            | top<br>middle<br>bottom<br>left<br>center<br>right | transitional,<br>frameset         | nein                 | richtet die Grafik aus                                              |
| alt              | #CDATA                                             | strict, transitional,<br>frameset | ja                   | bestimmt einen Alternativtext, falls die Grafik nicht anzeigbar ist |
| border           | #CDATA                                             | transitional,<br>frameset         | nein                 | bewirkt einen Rahmen um die Grafik – Dicke in Pixeln oder Prozent   |
| height           | #CDATA                                             | strict, transitional,<br>frameset | nein                 | gibt die Anzeighöhe der Grafik in Pixeln oder Prozent an            |



| <i>Bedeutung</i>     | <i>Referenziert eine Grafik</i> |                                |      |                                                                                                |
|----------------------|---------------------------------|--------------------------------|------|------------------------------------------------------------------------------------------------|
| hspace               | #CDATA                          | transitional, frameset         | nein | bestimmt den horizontalen Abstand zwischen Grafik und umfließendem Text in Pixeln oder Prozent |
| ismap                | ismap<br>(leeres Attribut)      | strict, transitional, frameset | nein | Grafik benutzt ein serverseitiges Handling für verweissensitive Flächen                        |
| longdesc             | #CDATA                          | strict, transitional, frameset | nein | gibt einen URI (nach RFC 2396) an, wo eine Langbeschreibung der Grafik zu finden ist           |
| name                 | #CDATA                          | strict, transitional, frameset | nein | bestimmt einen Namen für die Grafik                                                            |
| src                  | #CDATA                          | strict, transitional, frameset | ja   | gibt die Quelldatei der Grafik an                                                              |
| usemap               | #CDATA                          | strict, transitional, frameset | nein | gibt den URI/den Ankernamen eines map-Elements an                                              |
| vspace               | #CDATA                          | transitional, frameset         | nein | bestimmt den horizontalen Abstand zwischen Grafik und umfließendem Text in Pixeln oder Prozent |
| width                | #CDATA                          | strict, transitional, frameset | nein | gibt die Anzeigebreite der Grafik an                                                           |
| [Universalattribute] |                                 |                                |      |                                                                                                |

## input

| <i>Bedeutung</i> | <i>Erzeugt ein Formularelement, Typ wird durch Attribut bestimmt</i> |                                |                      |                                                                                      |
|------------------|----------------------------------------------------------------------|--------------------------------|----------------------|--------------------------------------------------------------------------------------|
| <i>Attribut</i>  | <i>Wert</i>                                                          | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                     |
| accept           | #CDATA                                                               | strict, transitional, frameset | nein                 | Listet bei input type="file" erlaubte Mime-Typen für File-Upload (nach RFC 2045) auf |
| accesskey        | #CDATA                                                               | strict, transitional, frameset | nein                 | bestimmt ein einzelnes Zeichen als Hotkey für das Element                            |
| align            | top<br>middle<br>bottom<br>left<br>center<br>right                   | transitional, frameset         | nein                 | richtet das Element aus                                                              |
| alt              | #CDATA                                                               | strict, transitional, frameset | nein                 | bestimmt einen Alternativtext, falls das Element nicht anzeigbar ist                 |
| checked          | checked<br>(leeres Attribut)                                         | strict, transitional, frameset | nein                 | bestimmt bei type="radio" oder type="checkbox", dass das Element ausgewählt ist      |

| <i>Bedeutung</i> | <i>Erzeugt ein Formularelement, Typ wird durch Attribut bestimmt</i>                          |                                |                           |                                                                                                                                                                                                  |
|------------------|-----------------------------------------------------------------------------------------------|--------------------------------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| datafld          | #CDATA                                                                                        | strict, transitional, frameset | nein                      | gibt den Namen eines Datenfeldes des aktuellen Datensatzes der mit <code>datasrc=</code> bestimmten Datenquelle an                                                                               |
| datasrc          | #CDATA                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| dataformatas     | plain-text<br>html                                                                            | strict, transitional, frameset | nein (Default: plaintext) |                                                                                                                                                                                                  |
| disabled         | disabled<br>(leeres Attribut)                                                                 | strict, transitional, frameset | nein                      | gibt an, wie die Daten, die aufgrund der Angaben <code>datasrc=</code> und <code>datafld=</code> ermittelt werden, zu interpretieren sind bestimmt, dass das Element nicht verändert werden kann |
| ismap            | ismap<br>(leeres Attribut)                                                                    | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| maxlength        | [Zahl]                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| name             | #CDATA                                                                                        | strict, transitional, frameset | nein                      | bestimmt einen Namen für das Element                                                                                                                                                             |
| onblur           | #CDATA                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| onchange         | #CDATA                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| onfocus          | #CDATA                                                                                        | strict, transitional, frameset | nein                      | bestimmt die maximale Anzahl erlaubter Zeichen im Element                                                                                                                                        |
| onselect         | #CDATA                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| readonly         | readonly<br>(leeres Attribut)                                                                 | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| size             | #CDATA                                                                                        | strict, transitional, frameset | nein                      | bestimmt die Anzeigebreite des Elements                                                                                                                                                          |
| src              | #CDATA                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| tabindex         | [Zahl]                                                                                        | strict, transitional, frameset | nein                      |                                                                                                                                                                                                  |
| type             | text<br>password<br>checkbox<br>radio<br>submit<br>reset<br>file<br>hidden<br>image<br>button | strict, transitional, frameset | nein (Default: text)      | bestimmt den Typ des Formularelements                                                                                                                                                            |

| Bedeutung            | Erzeugt ein Formularelement, Typ wird durch Attribut bestimmt |                                |      |                                                                          |
|----------------------|---------------------------------------------------------------|--------------------------------|------|--------------------------------------------------------------------------|
| usemap               | #CDATA                                                        | strict, transitional, frameset | nein | gibt bei input type="image" den URI/den Ankernamen eines map-Elements an |
| value                | #CDATA                                                        | strict, transitional, frameset | nein | bestimmt den Wert des Elements                                           |
| [Universalattribute] |                                                               |                                |      |                                                                          |

**ins**

| Bedeutung            | Markiert Text als eingefügt im Sinne von Änderungsmarkierung |                                |               |                                                                            |
|----------------------|--------------------------------------------------------------|--------------------------------|---------------|----------------------------------------------------------------------------|
| Attribut             | Wert                                                         | HTML-Varianten                 | erforderlich? | Bedeutung                                                                  |
| cite                 | #CDATA                                                       | strict, transitional, frameset | nein          | gibt den URI (nach RFC 2396) an, wo Gründe für die Änderung angegeben sind |
| datetime             | #CDATA                                                       | strict, transitional, frameset | nein          | gibt Datum und Uhrzeit der Änderung nach ISO-Format an                     |
| [Universalattribute] |                                                              |                                |               |                                                                            |

**isindex**

| Bedeutung                                         | Erzeugt ein einzeliges Eingabefeld |                        |               |                                     |
|---------------------------------------------------|------------------------------------|------------------------|---------------|-------------------------------------|
| Attribut                                          | Wert                               | HTML-Varianten         | erforderlich? | Bedeutung                           |
| prompt                                            | #CDATA                             | transitional, frameset | nein          | bestimmt den Labeltext des Elements |
| Universalattribute: id class style title dir lang |                                    |                        |               |                                     |

**kbd**

| Bedeutung            | Markiert Text als Tastatureingabe |
|----------------------|-----------------------------------|
| [Universalattribute] |                                   |

**label**

| <i>Bedeutung</i>     |             | <i>Ordnet Text einem bestimmten Formularfeld zu</i> |                      |                                                                      |
|----------------------|-------------|-----------------------------------------------------|----------------------|----------------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i> | <i>HTML-Varianten</i>                               | <i>erforderlich?</i> | <i>Bedeutung</i>                                                     |
| accesskey            | #CDATA      | strict, transitional, frameset                      | nein                 | bestimmt ein einzelnes Zeichen als Hotkey für das zugehörige Element |
| for                  | [IDREF]     | strict, transitional, frameset                      | nein                 | gibt die ID des Elements an, zu dem das Label gehört                 |
| onblur               | #CDATA      | strict, transitional, frameset                      | nein                 | beim Verlassen des zugehörigen Elements... (Script-Code)             |
| onfocus              | #CDATA      | strict, transitional, frameset                      | nein                 | wenn das zugehörige Element den Fokus hat... (Script-Code)           |
| [Universalattribute] |             |                                                     |                      |                                                                      |

**legend**

| <i>Bedeutung</i>     |                                | <i>Definiert eine Überschrift für eine Gruppe von Formularelementen</i> |                      |                                                                 |
|----------------------|--------------------------------|-------------------------------------------------------------------------|----------------------|-----------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>                    | <i>HTML-Varianten</i>                                                   | <i>erforderlich?</i> | <i>Bedeutung</i>                                                |
| accesskey            | #CDATA                         | strict, transitional, frameset                                          | nein                 | bestimmt ein einzelnes Zeichen als Hotkey für die Elementgruppe |
| align                | left<br>top<br>right<br>bottom | transitional, frameset                                                  | nein                 | richtet den Inhalt der Überschrift aus                          |
| [Universalattribute] |                                |                                                                         |                      |                                                                 |

**li**

| <i>Bedeutung</i>     |                                                   | <i>Markiert einen Listeneintrag</i> |                      |                                                                              |
|----------------------|---------------------------------------------------|-------------------------------------|----------------------|------------------------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>                                       | <i>HTML-Varianten</i>               | <i>erforderlich?</i> | <i>Bedeutung</i>                                                             |
| type                 | disc<br>square<br>circle<br>1<br>a<br>A<br>i<br>I | transitional, frameset              | nein                 | bestimmt das Aufzählungszeichen oder den Nummerierungstyp des Listeneintrags |
| value                | [Zahl]                                            | transitional, frameset              | nein                 | setzt den Wert der laufenden Nummerierung neu                                |
| [Universalattribute] |                                                   |                                     |                      |                                                                              |

**link**

| <i>Bedeutung</i>     |                                              | <i>Definiert logische Beziehungen zu anderen Dateien</i> |                           |                                                              |
|----------------------|----------------------------------------------|----------------------------------------------------------|---------------------------|--------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>                                  | <i>HTML-Varianten</i>                                    | <i>erforderlich?</i>      | <i>Bedeutung</i>                                             |
| charset              | #CDATA                                       | strict, transitional, frameset                           | nein                      | gibt den Zeichensatz der Verknüpfung (laut RFC 2045) an      |
| href                 | #CDATA                                       | strict, transitional, frameset                           | nein                      | gibt den URI der Verknüpfung (nach RFC 2396) an              |
| hreflang             | #CDATA                                       | strict, transitional, frameset                           | nein                      | gibt die Landessprache der Verknüpfung (nach RFC 1766) an    |
| media                | #CDATA                                       | strict, transitional, frameset                           | nein<br>(Default: screen) | bestimmt das Ausgabemedium                                   |
| rel                  | #CDATA                                       | strict, transitional, frameset                           | nein                      | bestimmt die Verknüpfung als »Vorwärtsverknüpfung«           |
| rev                  | #CDATA                                       | strict, transitional, frameset                           | nein                      | bestimmt die Verknüpfung als »Rückwärtsverknüpfung«          |
| target               | #CDATA<br>_blank<br>_parent<br>_self<br>_top | strict, transitional, frameset                           | nein                      | bestimmt den Fensternamen zur Ausgabe des Verknüpfungsziels  |
| type                 | #CDATA                                       | strict, transitional, frameset                           | nein                      | bestimmt den Mime-Type des Verknüpfungsziels (laut RFC 2045) |
| [Universalattribute] |                                              |                                                          |                           |                                                              |

**map**

| <i>Bedeutung</i>     |             | <i>Markiert einen Definitionsbereich für verweissensitive Grafiken</i> |                      |                                           |
|----------------------|-------------|------------------------------------------------------------------------|----------------------|-------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i> | <i>HTML-Varianten</i>                                                  | <i>erforderlich?</i> | <i>Bedeutung</i>                          |
| name                 | #CDATA      | strict, transitional, frameset                                         | ja                   | bestimmt einen Ankernamen für den Bereich |
| [Universalattribute] |             |                                                                        |                      |                                           |

**menu**

| <i>Bedeutung</i>     |                              | <i>Markiert eine Menüliste</i> |                      |                                                              |
|----------------------|------------------------------|--------------------------------|----------------------|--------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>                  | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                             |
| compact              | compact<br>(leeres Attribut) | transitional, frameset         | nein                 | bestimmt, dass die Liste in engerer Schrift dargestellt wird |
| [Universalattribute] |                              |                                |                      |                                                              |

**meta**

| <i>Bedeutung</i> <i>Definiert Meta-Daten wie Kurzbeschreibung und Stichwörter für eine Datei</i> |             |                                |                      |                                                                                       |
|--------------------------------------------------------------------------------------------------|-------------|--------------------------------|----------------------|---------------------------------------------------------------------------------------|
| <i>Attribut</i>                                                                                  | <i>Wert</i> | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                      |
| name                                                                                             | [Name]      | strict, transitional, frameset | nein                 | benennt eine bestimmte Eigenschaft für Meta-Informationen                             |
| content                                                                                          | #CDATA      | strict, transitional, frameset | ja                   | definiert den Wert für die Eigenschaft, die bei http-equiv= oder name= definiert wird |
| http-equiv                                                                                       | [Name]      | strict, transitional, frameset | nein                 | benennt eine bestimmte Eigenschaft für Meta-Informationen (für HTTP-Kopfdaten)        |
| scheme                                                                                           | [Name]      | strict, transitional, frameset | nein                 | bestimmt ein Schema für den Wert der Eigenschaft                                      |
| Universalattribute: dir   lang                                                                   |             |                                |                      |                                                                                       |

**noframes**

| <i>Bedeutung</i> <i>Markiert einen Bereich für den Fall, dass keine Frames angezeigt werden können</i> |  |
|--------------------------------------------------------------------------------------------------------|--|
| [Universalattribute]                                                                                   |  |

**noscript**

| <i>Bedeutung</i> <i>Markiert einen Bereich für den Fall, dass ein Script nicht ausgeführt werden kann</i> |  |
|-----------------------------------------------------------------------------------------------------------|--|
| [Universalattribute]                                                                                      |  |

**object**

| <i>Bedeutung</i> <i>Definiert ein Multimedia-Objekt</i> |                                                    |                                |                      |                                                                     |
|---------------------------------------------------------|----------------------------------------------------|--------------------------------|----------------------|---------------------------------------------------------------------|
| <i>Attribut</i>                                         | <i>Wert</i>                                        | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                                    |
| align                                                   | top<br>middle<br>bottom<br>left<br>center<br>right | transitional, frameset         | nein                 | richtet das Objekt aus                                              |
| archive                                                 | #CDATA                                             | strict, transitional, frameset | nein                 | gibt eine kommaseparierte Liste mit Archivdateien für das Objekt an |

| Bedeutung            | Definiert ein Multimedia-Objekt |                                   |                                 |                                                                                                                       |
|----------------------|---------------------------------|-----------------------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| border               | #CDATA                          | transitional,<br>frameset         | nein                            | bewirkt einen Rahmen um das Objekt<br>– Dicke in Pixeln oder Prozent                                                  |
| classid              | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt einen URI (nach RFC 2396) oder<br>eine eindeutige Bezeichnung für das<br>Objekt an                               |
| codebase             | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt den Basis-URI für Angaben zu<br>classid=, data= und archive= (nach<br>RFC 2396) an                               |
| codetype             | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt den Mime-Type des Quelltextes<br>der Datenquelle (nach RFC 2045) an                                              |
| data                 | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt den URI der Datenquelle (nach<br>RFC 2396) an                                                                    |
| datafld              | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt den Namen eines Datenfeldes des<br>aktuellen Datensatzes der mit<br>datasrc= bestimmten Datenquelle an           |
| datasrc              | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt den URI einer Datenquelle (nach<br>RFC 2396) an, die mit dem Objekt ver-<br>knüpft ist                           |
| dataformatas         | plaintext<br>html               | strict, transitional,<br>frameset | nein<br>(Default:<br>plaintext) | gibt an, wie die Daten, die aufgrund<br>der Angaben datasrc= und datafld=<br>ermittelt werden, zu interpretieren sind |
| declare              | declare<br>(leeres<br>Attribut) | strict, transitional,<br>frameset | nein                            | bewirkt, dass das Objekt zwar geladen,<br>aber nicht initialisiert (z.B. nicht ab-<br>gespielt) wird                  |
| height               | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt die Anzeighöhe des Objekts in Pi-<br>xeln oder Prozent an                                                        |
| hspace               | #CDATA                          | transitional,<br>frameset         | nein                            | bestimmt den horizontalen Abstand<br>zwischen Objekt und umfließendem<br>Text in Pixeln oder Prozent                  |
| name                 | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt einen Namen für das Objekt an                                                                                    |
| standby              | #CDATA                          | strict, transitional,<br>frameset | nein                            | definiert einen Meldungstext, der an-<br>gezeigt wird, während das Objekt gela-<br>den wird                           |
| tabindex             | [Zahl]                          | strict, transitional,<br>frameset | nein                            | bestimmt die Tabulatorreihenfolge für<br>Objekte durch Angabe einer Zahl                                              |
| type                 | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt den Mime-Type der Datenquelle<br>(nach RFC 2045) an                                                              |
| usemap               | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt bei Objekten, die verweissensitive<br>Grafiken behandeln, den URI/den An-<br>kernamen eines map-Elements an      |
| vspace               | #CDATA                          | transitional,<br>frameset         | nein                            | bestimmt den horizontalen Abstand<br>zwischen Objekt und umfließendem<br>Text in Pixeln oder Prozent                  |
| width                | #CDATA                          | strict, transitional,<br>frameset | nein                            | gibt die Anzeigebreite des Objekts in<br>Pixeln oder Prozent an                                                       |
| [Universalattribute] |                                 |                                   |                                 |                                                                                                                       |

**ol**

| <i>Bedeutung</i> <i>Markiert eine nummerierte Liste</i> |                              |                           |                      |                                                              |
|---------------------------------------------------------|------------------------------|---------------------------|----------------------|--------------------------------------------------------------|
| <i>Attribut</i>                                         | <i>Wert</i>                  | <i>HTML-Varianten</i>     | <i>erforderlich?</i> | <i>Bedeutung</i>                                             |
| compact                                                 | compact<br>(leeres Attribut) | transitional,<br>frameset | nein                 | bestimmt, dass die Liste in engerer Schrift dargestellt wird |
| start                                                   | [Zahl]                       | transitional,<br>frameset | nein                 | bestimmt einen Startwert für die Nummerierung (Dezimalwert)  |
| type                                                    | 1<br>a<br>A<br>i<br>I        | transitional,<br>frameset | nein                 | bestimmt die Art der Nummerierung                            |
| [Universalattribute]                                    |                              |                           |                      |                                                              |

**optgroup**

| <i>Bedeutung</i> <i>Markiert eine Gruppe von Einträgen in einer Menüstruktur</i> |                               |                                   |                      |                                                                      |
|----------------------------------------------------------------------------------|-------------------------------|-----------------------------------|----------------------|----------------------------------------------------------------------|
| <i>Attribut</i>                                                                  | <i>Wert</i>                   | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                     |
| disabled                                                                         | disabled<br>(leeres Attribut) | strict, transitional,<br>frameset | nein                 | bestimmt, dass die Gruppe von Einträgen nicht ausgewählt werden kann |
| label                                                                            | #CDATA                        | strict, transitional,<br>frameset | ja                   | bestimmt den Text des Gruppeneintrags                                |
| [Universalattribute]                                                             |                               |                                   |                      |                                                                      |

**option**

| <i>Bedeutung</i> <i>Markiert einen Eintrag in einer Auswahlliste</i> |                               |                                   |                      |                                                                   |
|----------------------------------------------------------------------|-------------------------------|-----------------------------------|----------------------|-------------------------------------------------------------------|
| <i>Attribut</i>                                                      | <i>Wert</i>                   | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                  |
| disabled                                                             | disabled<br>(leeres Attribut) | strict, transitional,<br>frameset | nein                 | bestimmt, dass der Eintrag nicht ausgewählt werden kann           |
| label                                                                | #CDATA                        | strict, transitional,<br>frameset | nein                 | bestimmt den Text des Eintrags im Zusammenhang mit Menüstrukturen |
| selected                                                             | selected<br>(leeres Attribut) | strict, transitional,<br>frameset | nein                 | bestimmt, dass vorausgewählt ist                                  |
| value                                                                | #CDATA                        | strict, transitional,<br>frameset | nein                 | bestimmt den Absendewert des Eintrags                             |
| [Universalattribute]                                                 |                               |                                   |                      |                                                                   |



**p**

| <i>Bedeutung</i> <i>Markiert einen Textabsatz</i> |                                    |                           |                            |                                         |
|---------------------------------------------------|------------------------------------|---------------------------|----------------------------|-----------------------------------------|
| <i>Attribut</i>                                   | <i>Wert</i>                        | <i>HTML-Varianten</i>     | <i>erforderlich?</i>       | <i>Bedeutung</i>                        |
| align                                             | left<br>center<br>right<br>justify | transitional,<br>frameset | nein<br>(Default:<br>left) | richtet den Inhalt des Textabsatzes aus |
| [Universalattribute]                              |                                    |                           |                            |                                         |

**param**

| <i>Bedeutung</i> <i>Definiert Parameter eines Multimedia-Objekts oder Java-Applets</i> |                       |                                   |                            |                                                                              |
|----------------------------------------------------------------------------------------|-----------------------|-----------------------------------|----------------------------|------------------------------------------------------------------------------|
| <i>Attribut</i>                                                                        | <i>Wert</i>           | <i>HTML-Varianten</i>             | <i>erforderlich?</i>       | <i>Bedeutung</i>                                                             |
| id                                                                                     | [ID]                  | strict, transitional,<br>frameset | nein                       | gibt eine eindeutige Bezeichnung für den Parameter an                        |
| name                                                                                   | #CDATA                | strict, transitional,<br>frameset | ja                         | gibt einen Namen für den Parameter an                                        |
| value                                                                                  | #CDATA                | strict, transitional,<br>frameset | nein                       | gibt einen Initialisierungswert für den Parameter an                         |
| valuetype                                                                              | data<br>ref<br>object | strict, transitional,<br>frameset | nein<br>(Default:<br>data) | gibt an, wie der Initialisierungswert von value= zu interpretieren ist       |
| type                                                                                   | #CDATA                | strict, transitional,<br>frameset | nein                       | gibt den Mime-Type des Wertes (nach RFC 2045) im Fall von valuetype="ref" an |

**pre**

| <i>Bedeutung</i> <i>Formatiert einen Bereich wie im Editor eingegeben (präformatierter Text)</i> |             |                           |                      |                                       |
|--------------------------------------------------------------------------------------------------|-------------|---------------------------|----------------------|---------------------------------------|
| <i>Attribut</i>                                                                                  | <i>Wert</i> | <i>HTML-Varianten</i>     | <i>erforderlich?</i> | <i>Bedeutung</i>                      |
| width                                                                                            | [Zahl]      | transitional,<br>frameset | nein                 | gibt eine Anzahl Zeichen pro Zeile an |
| [Universalattribute]                                                                             |             |                           |                      |                                       |

**q**

| <i>Bedeutung</i>     |                     | <i>Markiert ein Zitat</i>      |                      |                                                 |
|----------------------|---------------------|--------------------------------|----------------------|-------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>         | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                |
| <code>cite</code>    | <code>#CDATA</code> | strict, transitional, frameset | nein                 | gibt den URI der Zitatquelle (nach RFC 2396) an |
| [Universalattribute] |                     |                                |                      |                                                 |

**s**

| <i>Bedeutung</i>     |  | <i>Markiert Text als durchgestrichen</i> |  |  |
|----------------------|--|------------------------------------------|--|--|
| [Universalattribute] |  |                                          |  |  |

**samp**

| <i>Bedeutung</i>     |  | <i>Markiert Text als Beispiel</i> |  |  |
|----------------------|--|-----------------------------------|--|--|
| [Universalattribute] |  |                                   |  |  |

**script**

| <i>Bedeutung</i>      |                                         | <i>Definiert einen Bereich für Scripts (z.B. JavaScript)</i> |                      |                                                                                                      |
|-----------------------|-----------------------------------------|--------------------------------------------------------------|----------------------|------------------------------------------------------------------------------------------------------|
| <i>Attribut</i>       | <i>Wert</i>                             | <i>HTML-Varianten</i>                                        | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                                     |
| <code>charset</code>  | <code>#CDATA</code>                     | strict, transitional, frameset                               | nein                 | gibt den Zeichensatz einer mit <code>src=</code> angegebenen externen Scriptdatei (laut RFC 2045) an |
| <code>defer</code>    | <code>defer</code><br>(leeres Attribut) | strict, transitional, frameset                               | nein                 | verhindert, dass das Script irgendwelche Inhalte im Dokumentfenster erzeugen kann                    |
| <code>event</code>    | <code>#CDATA</code>                     | strict, transitional, frameset                               | nein                 | führt das Script abhängig davon aus, ob ein bestimmtes Ereignis eingetreten ist                      |
| <code>language</code> | <code>#CDATA</code>                     | transitional, frameset                                       | nein                 | gibt die verwendete Script-Sprache an                                                                |
| <code>for</code>      | <code>[IDREF]</code>                    | strict, transitional, frameset                               | nein                 | gibt die ID des Elements an, für das das Script gelten soll                                          |
| <code>src</code>      | <code>#CDATA</code>                     | strict, transitional, frameset                               | nein                 | bestimmt einen URI (nach RFC 2396), der den Script-Code enthält                                      |
| <code>type</code>     | <code>#CDATA</code>                     | strict, transitional, frameset                               | ja                   | bestimmt den Mime-Type des Scripts (laut RFC 2045)                                                   |

**select**

| <i>Bedeutung</i>     |                               | <i>Definiert eine Auswahlliste</i> |                              |  | <i>Bedeutung</i>                                                                                                |
|----------------------|-------------------------------|------------------------------------|------------------------------|--|-----------------------------------------------------------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>                   | <i>HTML-Varianten</i>              | <i>erforderlich?</i>         |  |                                                                                                                 |
| datafld              | #CDATA                        | strict, transitional, frameset     | nein                         |  | gibt den Namen eines Datenfeldes des aktuellen Datensatzes der mit                                              |
| datasrc              | #CDATA                        | strict, transitional, frameset     | nein                         |  | datasrc= bestimmten Datenquelle an                                                                              |
| dataformatas         | plain-text<br>html            | strict, transitional, frameset     | nein<br>(Default: plaintext) |  | gibt den URI einer Datenquelle (nach RFC 2396) an, die mit der Auswahlliste verknüpft ist                       |
| disabled             | disabled<br>(leeres Attribut) | strict, transitional, frameset     | nein                         |  | gibt an, wie die Daten, die aufgrund der Angaben datasrc= und datafld= ermittelt werden, zu interpretieren sind |
| multiple             | multiple<br>(leeres Attribut) | strict, transitional, frameset     | nein                         |  | bestimmt, dass die Auswahlliste nicht benutzt werden kann                                                       |
| name                 | #CDATA                        | strict, transitional, frameset     | nein                         |  | erlaubt es, mehrere Einträge zu selektieren                                                                     |
| onblur               | #CDATA                        | strict, transitional, frameset     | nein                         |  | bestimmt einen Namen für die Auswahlliste                                                                       |
| onchange             | #CDATA                        | strict, transitional, frameset     | nein                         |  | beim Verlassen der Auswahlliste... (Script-Code)                                                                |
| onfocus              | #CDATA                        | strict, transitional, frameset     | nein                         |  | wenn ein (anderer) Eintrag ausgewählt wurde... (Script-Code)                                                    |
| size                 | #CDATA                        | strict, transitional, frameset     | nein                         |  | wenn die Auswahlliste den Fokus hat... (Script-Code)                                                            |
| tabindex             | [Zahl]                        | strict, transitional, frameset     | nein                         |  | bestimmt die Anzahl der gleichzeitig sichtbaren Einträge                                                        |
|                      |                               |                                    |                              |  | bestimmt die Tabulatorreihenfolge für Formularelemente durch Angabe einer Zahl                                  |
| [Universalattribute] |                               |                                    |                              |  |                                                                                                                 |

**small**

| <i>Bedeutung</i>     | <i>Markiert Text, der kleiner dargestellt wird als normal</i> |
|----------------------|---------------------------------------------------------------|
| [Universalattribute] |                                                               |

**span**

| <i>Bedeutung</i> <i>Markiert einen inline-Bereich zur Formatierung, z.B. mit CSS-Stylesheets</i> |                   |                                |                              |                                                                                                                  |
|--------------------------------------------------------------------------------------------------|-------------------|--------------------------------|------------------------------|------------------------------------------------------------------------------------------------------------------|
| <i>Attribut</i>                                                                                  | <i>Wert</i>       | <i>HTML-Varianten</i>          | <i>erforderlich?</i>         | <i>Bedeutung</i>                                                                                                 |
| datafld                                                                                          | #CDATA            | strict, transitional, frameset | nein                         | gibt den Namen eines Datenfeldes des aktuellen Datensatzes der mit data-src= bestimmten Datenquelle an           |
| data-src                                                                                         | #CDATA            | strict, transitional, frameset | nein                         | gibt den URI einer Datenquelle (nach RFC 2396) an, die mit dem Bereich verknüpft ist                             |
| data-formatas                                                                                    | plaintext<br>html | strict, transitional, frameset | nein<br>(Default: plaintext) | gibt an, wie die Daten, die aufgrund der Angaben data-src= und datafld= ermittelt werden, zu interpretieren sind |
| [Universalattribute]                                                                             |                   |                                |                              |                                                                                                                  |

**strike**

| <i>Bedeutung</i> <i>Markiert einen Text als durchgestrichen</i> |  |
|-----------------------------------------------------------------|--|
| [Universalattribute]                                            |  |

**strong**

| <i>Bedeutung</i> <i>Markiert einen Text als betont</i> |  |
|--------------------------------------------------------|--|
| [Universalattribute]                                   |  |

**style**

| <i>Bedeutung</i> <i>Markiert einen Bereich für Stylesheet-Definitionen, z.B. in CSS</i> |             |                                |                           |                                                          |
|-----------------------------------------------------------------------------------------|-------------|--------------------------------|---------------------------|----------------------------------------------------------|
| <i>Attribut</i>                                                                         | <i>Wert</i> | <i>HTML-Varianten</i>          | <i>erforderlich?</i>      | <i>Bedeutung</i>                                         |
| media                                                                                   | #CDATA      | strict, transitional, frameset | nein<br>(Default: screen) | bestimmt das Ausgabemedium                               |
| title                                                                                   | #CDATA      | strict, transitional, frameset | nein                      | bestimmt einen Titel für den Style-Bereich               |
| type                                                                                    | #CDATA      | strict, transitional, frameset | ja                        | bestimmt den Mime-Type der Style-Sprache (laut RFC 2045) |
| Universalattribute: dir   lang                                                          |             |                                |                           |                                                          |

**sub**

|                  |                                             |
|------------------|---------------------------------------------|
| <i>Bedeutung</i> | <i>Markiert einen Text als tiefgestellt</i> |
|------------------|---------------------------------------------|

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**sup**

|                  |                                             |
|------------------|---------------------------------------------|
| <i>Bedeutung</i> | <i>Markiert einen Text als hochgestellt</i> |
|------------------|---------------------------------------------|

|                      |
|----------------------|
| [Universalattribute] |
|----------------------|

**table**

|                  |                             |
|------------------|-----------------------------|
| <i>Bedeutung</i> | <i>Erzeugt eine Tabelle</i> |
|------------------|-----------------------------|

| <i>Attribut</i> | <i>Wert</i>             | <i>HTML-Varianten</i>             | <i>erforderlich?</i>            | <i>Bedeutung</i>                                                                                                                          |
|-----------------|-------------------------|-----------------------------------|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| align           | left<br>center<br>right | transitional,<br>frameset         | nein                            | richtet die Tabelle aus                                                                                                                   |
| border          | #CDATA                  | strict, transitional,<br>frameset | nein                            | bewirkt einen sichtbaren Rahmen um die Tabelle – Dicke in Pixeln                                                                          |
| bgcolor         | #CDATA                  | transitional,<br>frameset         | nein                            | gibt die tabellenweite Hintergrundfarbe hexadezimal (in der Form #XXXXXX) oder als Farbwort an                                            |
| cellpadding     | #CDATA                  | strict, transitional,<br>frameset | nein                            | bestimmt den Innenabstand zwischen Zellenrand und Zelleninhalt in Pixeln oder Prozent                                                     |
| cellspacing     | #CDATA                  | strict, transitional,<br>frameset | nein                            | bestimmt den Abstand zwischen den Zellen untereinander in Pixeln oder Prozent                                                             |
| datafld         | #CDATA                  | strict, transitional,<br>frameset | nein                            | gibt den Namen eines Datenfeldes des aktuellen Datensatzes der mit <code>datasrc=</code> bestimmten Datenquelle an                        |
| datasrc         | #CDATA                  | strict, transitional,<br>frameset | nein                            | gibt den URI einer Datenquelle (nach RFC 2396) an, die mit der Tabelle verknüpft ist                                                      |
| dataformatas    | plaint<br>ext<br>html   | strict, transitional,<br>frameset | nein<br>(Default:<br>plaintext) | gibt an, wie die Daten, die aufgrund der Angaben <code>datasrc=</code> und <code>datafld=</code> ermittelt werden, zu interpretieren sind |

| <i>Bedeutung</i>     | <i>Erzeugt eine Tabelle</i> |                                   |      |                                                                                          |
|----------------------|-----------------------------|-----------------------------------|------|------------------------------------------------------------------------------------------|
| frame                | void                        | strict, transitional,<br>frameset | nein | gibt an, welche Rahmen der Tabelle sichtbar sein sollen                                  |
|                      | above                       |                                   |      |                                                                                          |
|                      | below                       |                                   |      |                                                                                          |
|                      | hsides                      |                                   |      |                                                                                          |
|                      | lhs                         |                                   |      |                                                                                          |
|                      | rhs                         |                                   |      |                                                                                          |
|                      | vsides                      |                                   |      |                                                                                          |
|                      | box                         |                                   |      |                                                                                          |
|                      | border                      |                                   |      |                                                                                          |
|                      |                             |                                   |      |                                                                                          |
| rules                | none                        | strict, transitional,<br>frameset | nein | gibt an, welche Gitternetzlinien innerhalb der Tabelle sichtbar sein sollen              |
|                      | groups                      |                                   |      |                                                                                          |
|                      | rows                        |                                   |      |                                                                                          |
|                      | cols                        |                                   |      |                                                                                          |
|                      | all                         |                                   |      |                                                                                          |
| summary              | #CDATA                      | strict, transitional,<br>frameset | nein | definiert einen Text, der Sehbehinderten die Tabelle im Fall von Sprachausgabe erläutert |
| width                | #CDATA                      | strict, transitional,<br>frameset | nein | gibt die Anzeigebreite der Tabelle in Pixeln oder Prozent an                             |
| [Universalattribute] |                             |                                   |      |                                                                                          |

**tbody**

| Bedeutung            | Bestimmt in einer Tabelle den Tabellenkörper |                                |               |                                                                               |
|----------------------|----------------------------------------------|--------------------------------|---------------|-------------------------------------------------------------------------------|
| Attribut             | Wert                                         | HTML-Varianten                 | erforderlich? | Bedeutung                                                                     |
| align                | left                                         | strict, transitional, frameset | nein          | richtet die Tabellenzellen des Bereichs horizontal aus                        |
|                      | center                                       |                                |               |                                                                               |
|                      | right                                        |                                |               |                                                                               |
|                      | justify                                      |                                |               |                                                                               |
|                      | char                                         |                                |               |                                                                               |
| char                 | #CDATA                                       | strict, transitional, frameset | nein          | bestimmt in Verbindung mit align="char" das Ausrichtungszeichen               |
| charoff              | #CDATA                                       | strict, transitional, frameset | nein          | bestimmt in Verbindung mit align="char" die Position des Ausrichtungszeichens |
| valign               | top                                          | strict, transitional, frameset | nein          | richtet Tabellenzellen des Bereichs vertikal aus                              |
|                      | middle                                       |                                |               |                                                                               |
|                      | bottom                                       |                                |               |                                                                               |
|                      | baseline                                     |                                |               |                                                                               |
| [Universalattribute] |                                              |                                |               |                                                                               |

**td**

| <i>Bedeutung</i> |                                            | <i>Markiert eine Tabellenzelle</i> |                      |                                                                                                                                                              |  |
|------------------|--------------------------------------------|------------------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <i>Attribut</i>  | <i>Wert</i>                                | <i>HTML-Varianten</i>              | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                                                                                             |  |
| abbr             | #CDATA                                     | strict, transitional, frameset     | nein                 | definiert einen kurzen Hinweistext für die Zuordnung der Tabellenzelle, die z.B. bei der Sprachausgabe dem Zelleninhalt vorangesprochen wird                 |  |
| align            | left<br>center<br>right<br>justify<br>char | strict, transitional, frameset     | nein                 | richtet die Tabellenzelle horizontal aus                                                                                                                     |  |
| axis             | #CDATA                                     | strict, transitional, frameset     | nein                 | definiert eine Liste von Kategorien, zu der die Zelle gehört, und die z.B. bei der Sprachausgabe dem Zelleninhalt vorangesprochen werden                     |  |
| bgcolor          | #CDATA                                     | transitional, frameset             | nein                 | gibt die Hintergrundfarbe der Zelle hexadezimal (in der Form #XXXXXX) oder als Farbwert an                                                                   |  |
| char             | #CDATA                                     | strict, transitional, frameset     | nein                 | bestimmt in Verbindung mit <code>align="char"</code> das Ausrichtungszeichen                                                                                 |  |
| charoff          | #CDATA                                     | strict, transitional, frameset     | nein                 | bestimmt in Verbindung mit <code>align="char"</code> die Position des Ausrichtungszeichens                                                                   |  |
| colspan          | [Zahl]                                     | strict, transitional, frameset     | nein<br>(Default: 1) | bestimmt, über wie viele Spalten sich die aktuelle Zelle erstrecken soll                                                                                     |  |
| headers          | [IDREFS]                                   | strict, transitional, frameset     | nein                 | definiert eine Liste von ID-Namen von Zeilen- oder Spaltenüberschriften, zu denen die Zelle gehört, was z.B. für Sprachausgabe der Tabelle von Bedeutung ist |  |
| height           | #CDATA                                     | transitional, frameset             | nein                 | gibt die Anzeighöhe der Zelle in Pixeln oder Prozent an                                                                                                      |  |
| nowrap           | nowrap<br>(leeres Attribut)                | transitional, frameset             | nein                 | verhindert automatischen Zeilenumbruch innerhalb der Zelle                                                                                                   |  |
| rowspan          | [Zahl]                                     | strict, transitional, frameset     | nein<br>(Default: 1) | bestimmt, über wie viele Zeilen sich die aktuelle Zelle erstrecken soll                                                                                      |  |
| scope -          | row<br>col<br>rowgroup<br>colgroup         | strict, transitional, frameset     | nein                 | definiert, für welche restlichen Zeilen oder Spalten die aktuelle Zelle Überschriftencharakter hat, was z.B. für Sprachausgabe der Tabelle von Bedeutung ist |  |

| <i>Bedeutung</i>     | <i>Markiert eine Tabellenzelle</i>  |                           |      |                                                               |
|----------------------|-------------------------------------|---------------------------|------|---------------------------------------------------------------|
| <code>valign</code>  | top<br>middle<br>bottom<br>baseline | transitional,<br>frameset | nein | richtet Tabellenzelle vertikal aus                            |
| <code>width</code>   | <code>#CDATA</code>                 | transitional,<br>frameset | nein | gibt die Anzeigebreite der Zelle in<br>Pixeln oder Prozent an |
| [Universalattribute] |                                     |                           |      |                                                               |

**textarea**

| <i>Bedeutung</i>          | <i>Definiert ein mehrzeiliges Eingabefeld</i> |                                   |                                 |                                                                                                                                                 |
|---------------------------|-----------------------------------------------|-----------------------------------|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Attribut</i>           | <i>Wert</i>                                   | <i>HTML-Varianten</i>             | <i>erforderlich?</i>            | <i>Bedeutung</i>                                                                                                                                |
| <code>accesskey</code>    | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | bestimmt ein einzelnes Zeichen als<br>Hotkey für das Element                                                                                    |
| <code>cols</code>         | [Zahl]                                        | strict, transitional,<br>frameset | ja                              | bestimmt die Anzahl Zeichen pro Zeile                                                                                                           |
| <code>datafld</code>      | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | gibt den Namen eines Datenfeldes des<br>aktuellen Datensatzes der mit <code>data-</code><br><code>tasrc=</code> bestimmten Datenquelle an       |
| <code>datasrc</code>      | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | gibt den URI einer Datenquelle (nach<br>RFC 2396) an, die mit dem<br>Formularelement verknüpft ist                                              |
| <code>dataformatas</code> | plaintext<br>html                             | strict, transitional,<br>frameset | nein<br>(Default:<br>plaintext) | gibt an, wie die Daten, die aufgrund<br>der Angaben <code>datasrc=</code> und <code>datafld=</code><br>ermittelt werden, zu interpretieren sind |
| <code>disabled</code>     | disabled<br>(leeres<br>Attribut)              | strict, transitional,<br>frameset | nein                            | bestimmt, dass das Element nicht<br>verändert werden kann                                                                                       |
| <code>name</code>         | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | bestimmt einen Namen für das Ele-<br>ment                                                                                                       |
| <code>onblur</code>       | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | beim Verlassen des Elements... (Script-<br>Code)                                                                                                |
| <code>onchange</code>     | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | wenn das Element geändert wurde...<br>(Script-Code)                                                                                             |
| <code>onfocus</code>      | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | wenn das Element den Fokus hat...<br>(Script-Code)                                                                                              |
| <code>onselect</code>     | <code>#CDATA</code>                           | strict, transitional,<br>frameset | nein                            | wenn das Element ausgewählt ist...<br>(Script-Code)                                                                                             |
| <code>readonly</code>     | readonly<br>(leeres<br>Attribut)              | strict, transitional,<br>frameset | nein                            | bestimmt, dass der Wert des Elements<br>nicht verändert werden kann                                                                             |



| <i>Bedeutung</i>     | <i>Definiert ein mehrzeiliges Eingabefeld</i> |                                |      |                                                                                |
|----------------------|-----------------------------------------------|--------------------------------|------|--------------------------------------------------------------------------------|
| rows                 | [Zahl]                                        | strict, transitional, frameset | ja   | bestimmt die Anzahl sichtbarer Zeilen                                          |
| tabindex             | [Zahl]                                        | strict, transitional, frameset | nein | bestimmt die Tabulatorreihenfolge für Formularelemente durch Angabe einer Zahl |
| [Universalattribute] |                                               |                                |      |                                                                                |

**tfoot**

| <i>Bedeutung</i>     | <i>Bestimmt in einer Tabelle den Tabellenfuß</i> |                                |                      |                                                                               |
|----------------------|--------------------------------------------------|--------------------------------|----------------------|-------------------------------------------------------------------------------|
| <i>Attribut</i>      | <i>Wert</i>                                      | <i>HTML-Varianten</i>          | <i>erforderlich?</i> | <i>Bedeutung</i>                                                              |
| align                | left<br>center<br>right<br>justify<br>char       | strict, transitional, frameset | nein                 | richtet die Tabellenzellen des Bereichs horizontal aus                        |
| char                 | #CDATA                                           | strict, transitional, frameset | nein                 | bestimmt in Verbindung mit align="char" das Ausrichtungszeichen               |
| charoff              | #CDATA                                           | strict, transitional, frameset | nein                 | bestimmt in Verbindung mit align="char" die Position des Ausrichtungszeichens |
| valign               | top<br>middle<br>bottom<br>baseline              | strict, transitional, frameset | nein                 | richtet Tabellenzellen des Bereichs vertikal aus                              |
| [Universalattribute] |                                                  |                                |                      |                                                                               |

**th**

| <i>Bedeutung</i> | <i>Markiert eine Tabellenkopfzelle</i>     |                                |                      |                                                                                                                                              |
|------------------|--------------------------------------------|--------------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Attribut</i>  | <i>Wert</i>                                | <i>HTML-Variante</i>           | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                                                                             |
| abbr             | #CDATA                                     | strict, transitional, frameset | nein                 | definiert einen kurzen Hinweistext für die Zuordnung der Tabellenzelle, die z.B. bei der Sprachausgabe dem Zelleninhalt vorangesprochen wird |
| align            | left<br>center<br>right<br>justify<br>char | strict, transitional, frameset | nein                 | richtet die Tabellenkopfzelle horizontal aus                                                                                                 |

| Bedeutung            | Markiert eine Tabellenkopfzelle     |                                      |                   |                                                                                                                                                              |
|----------------------|-------------------------------------|--------------------------------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| axis                 | #CDATA                              | strict,<br>transitional,<br>frameset | nein              | definiert eine Liste von Kategorien, zu der die Zelle gehört und die z.B. bei der Sprachausgabe dem Zelleninhalt vorangesprochen werden                      |
| bgcolor              | #CDATA                              | transitional,<br>frameset            | nein              | gibt die Hintergrundfarbe der Zelle hexadezimal (in der Form #XXXXXX) oder als Farbwert an                                                                   |
| char                 | #CDATA                              | strict,<br>transitional,<br>frameset | nein              | bestimmt in Verbindung mit <code>align="char"</code> das Ausrichtungszeichen                                                                                 |
| charoff              | #CDATA                              | strict,<br>transitional,<br>frameset | nein              | bestimmt in Verbindung mit <code>align="char"</code> die Position des Ausrichtungszeichens                                                                   |
| colspan              | [Zahl]                              | strict,<br>transitional,<br>frameset | nein (Default: 1) | bestimmt, über wie viele Spalten sich die aktuelle Zelle erstrecken soll                                                                                     |
| headers              | [IDREFS]                            | strict,<br>transitional,<br>frameset | nein              | definiert eine Liste von ID-Namen von Zeilen- oder Spaltenüberschriften, zu denen die Zelle gehört, was z.B. für Sprachausgabe der Tabelle von Bedeutung ist |
| height               | #CDATA                              | transitional,<br>frameset            | nein              | gibt die Anzeigehöhe der Zelle in Pixeln oder Prozent an                                                                                                     |
| nowrap               | nowrap<br>(leeres Attribut)         | transitional,<br>frameset            | nein              | verhindert automatischen Zeilenumbruch innerhalb der Zelle                                                                                                   |
| rowspan              | [Zahl]                              | strict,<br>transitional,<br>frameset | nein (Default: 1) | bestimmt, über wie viele Zeilen sich die aktuelle Zelle erstrecken soll                                                                                      |
| scope                | row<br>col<br>rowgroup<br>colgroup  | strict,<br>transitional,<br>frameset | nein              | definiert, für welche restlichen Zeilen oder Spalten die aktuelle Zelle Überschriftencharakter hat, was z.B. für Sprachausgabe der Tabelle von Bedeutung ist |
| valign               | top<br>middle<br>bottom<br>baseline | strict,<br>transitional,<br>frameset | nein              | richtet Tabellenkopfzelle vertikal aus                                                                                                                       |
| width                | #CDATA                              | transitional,<br>frameset            | nein              | gibt die Anzeigebreite der Zelle in Pixeln oder Prozent an                                                                                                   |
| [Universalattribute] |                                     |                                      |                   |                                                                                                                                                              |

**thead**

| <i>Bedeutung</i> <i>Bestimmt in einer Tabelle den Tabellenkopf</i> |                                            |                                   |                      |                                                                               |
|--------------------------------------------------------------------|--------------------------------------------|-----------------------------------|----------------------|-------------------------------------------------------------------------------|
| <i>Attribut</i>                                                    | <i>Wert</i>                                | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                              |
| align                                                              | left<br>center<br>right<br>justify<br>char | strict, transitional,<br>frameset | nein                 | richtet die Tabellenzellen des Bereichs horizontal aus                        |
| char                                                               | #CDATA                                     | strict, transitional,<br>frameset | nein                 | bestimmt in Verbindung mit align="char" das Ausrichtungszeichen               |
| charoff                                                            | #CDATA                                     | strict, transitional,<br>frameset | nein                 | bestimmt in Verbindung mit align="char" die Position des Ausrichtungszeichens |
| valign                                                             | top<br>middle<br>bottom<br>baseline        | strict, transitional,<br>frameset | nein                 | richtet Tabellenzellen des Bereichs vertikal aus                              |
| [Universalattribute]                                               |                                            |                                   |                      |                                                                               |

**title**

| <i>Bedeutung</i>     | <i>Definiert den Titel einer HTML-Datei</i> |
|----------------------|---------------------------------------------|
| [Universalattribute] |                                             |

**tr**

| <i>Bedeutung</i> <i>Definiert eine Tabellenzeile</i> |                                            |                                   |                      |                                                                                                    |
|------------------------------------------------------|--------------------------------------------|-----------------------------------|----------------------|----------------------------------------------------------------------------------------------------|
| <i>Attribut</i>                                      | <i>Wert</i>                                | <i>HTML-Varianten</i>             | <i>erforderlich?</i> | <i>Bedeutung</i>                                                                                   |
| align                                                | left<br>center<br>right<br>justify<br>char | strict, transitional,<br>frameset | nein                 | richtet die Tabellenzellen der Zeile horizontal aus                                                |
| bgcolor                                              | #CDATA                                     | transitional,<br>frameset         | nein                 | gibt die Hintergrundfarbe der Tabellenzeile hexadezimal (in der Form #XXXXXX) oder als Farbwert an |
| char                                                 | #CDATA                                     | strict, transitional,<br>frameset | nein                 | bestimmt in Verbindung mit align="char" das Ausrichtungszeichen                                    |

## 1262 Anhang C: HTML: Attributreferenz

| <i>Bedeutung</i> <i>Definiert eine Tabellenzeile</i> |                                     |                                |      |                                                                               |
|------------------------------------------------------|-------------------------------------|--------------------------------|------|-------------------------------------------------------------------------------|
| charoff                                              | #CDATA                              | strict, transitional, frameset | nein | bestimmt in Verbindung mit align="char" die Position des Ausrichtungszeichens |
| valign                                               | top<br>middle<br>bottom<br>baseline | strict, transitional, frameset | nein | richtet Tabellenzellen der Zeile vertikal aus                                 |
| [Universalattribute]                                 |                                     |                                |      |                                                                               |

### ***tt***

| <i>Bedeutung</i> <i>Markiert einen Text, der im Schreibmaschinenstil dargestellt wird</i> |  |
|-------------------------------------------------------------------------------------------|--|
| [Universalattribute]                                                                      |  |

### ***u***

| <i>Bedeutung</i> <i>Markiert einen Text als unterstrichen</i> |  |
|---------------------------------------------------------------|--|
| [Universalattribute]                                          |  |

### ***ul***

| <i>Bedeutung</i> <i>Markiert eine Aufzählungsliste</i> |                              |                        |                      |                                                              |
|--------------------------------------------------------|------------------------------|------------------------|----------------------|--------------------------------------------------------------|
| <i>Attribut</i>                                        | <i>Wert</i>                  | <i>HTML-Varianten</i>  | <i>erforderlich?</i> | <i>Bedeutung</i>                                             |
| compact                                                | compact<br>(leeres Attribut) | transitional, frameset | nein                 | bestimmt, dass die Liste in engerer Schrift dargestellt wird |
| type                                                   | disc<br>square<br>circle     | transitional, frameset | nein                 | bestimmt die Art des Aufzählungszeichens                     |
| [Universalattribute]                                   |                              |                        |                      |                                                              |

### ***var***

| <i>Bedeutung</i> <i>Markiert einen Text als Variable</i> |  |
|----------------------------------------------------------|--|
| [Universalattribute]                                     |  |

## D HTML: Zeichenreferenz

### D.1 Allgemeines zur HTML-Zeichenreferenz

In der Zeichenreferenz geht es um Zeichen, die zwischen dem Anfangs- und dem End-Tag eines HTML-Elements als Text stehen können (siehe auch #PCDATA, Anhang B.4), sowie um Zeichen, die bei Wertzuweisungen an viele Attribute notiert werden können (siehe auch #CDATA, Anhang C.2). Bei einem HTML-Konstrukt wie `<p class="Aussage">Zeichen können viel bedeuten</p>` ist zum einen die Zeichenfolge `Zeichen können viel bedeuten` relevant und zum anderen die Zeichenfolge `Aussage`.

Seit HTML 4.0 kann jedes Zeichen aus dem Zeichenvorrat nach ISO 10646 (deckungsgleich mit dem Unicode-Standard) notiert werden. Beliebige Zeichen aus diesem gewaltigen Zeichenvorrat können durch eine spezielle numerische Notation erzeugt werden. Das Notations-Schema für solche numerischen Notationen lautet:

```
&#[x][Nummer];
```

Beispiele einer Notation in HTML:

```
&#945;  
&#x3B1;
```

Das `x` notieren Sie dann, wenn Sie die Nummer hexadezimal angeben. Bei dezimaler Angabe lassen Sie das `x` weg. Die Nummer eines gewünschten Zeichens können Sie aus den so genannten Unicode-Charts ermitteln (siehe Abschnitt 23.3 über das Unicode-System).

Neben der numerischen Unicode-Notation gibt es in HTML für häufiger verwendete Sonderzeichen auch so genannte **named entities**, benannte Zeichen. Alle benannten Zeichen werden nach dem folgenden Schema notiert:

```
&[Name];
```

Beispiel einer Notation in HTML:

```
&alpha;
```

Die verfügbaren benannten Zeichen können Sie den folgenden Tabellen entnehmen. In der jeweils ersten Spalte der Tabellen wird versucht, das Zeichen mit Hilfe des HTML-Namens darzustellen. Ein WWW-Browser muss jedoch HTML 4.0 interpretieren, und auch dann kann er nur solche Zeichen darstellen, für die er eine geeignete Schriftart vorfindet. In den Tabellen sind Zeichen, die im Buch aus technischen Gründen nicht gedruckt werden können, als *nicht darstellbar* markiert.

## D.2 Benannte Zeichen für HTML-eigene Zeichen

| Zeichen | Beschreibung                          | Name in HTML | Unicode in HTML |
|---------|---------------------------------------|--------------|-----------------|
| "       | Anführungszeichen oben                | &quot;       | &#34;           |
| &       | Ampersand-Zeichen, kaufmännisches Und | &amp;        | &#38;           |
| <       | öffnende spitze Klammer               | &lt;         | &#60;           |
| >       | schließende spitze Klammer            | &gt;         | &#62;           |

## D.3 Benannte Zeichen für den Zeichensatz ISO 8859-1

Die HTML-Namen dieser Zeichen sind seit HTML 3.2 verfügbar.

| Zeichen | Beschreibung                            | Name in HTML | Unicode in HTML |
|---------|-----------------------------------------|--------------|-----------------|
|         | erzwungenes Leerzeichen                 | &nbsp;       | &#160;          |
| !       | umgekehrtes Ausrufezeichen              | &iexcl;      | &#161;          |
| ¢       | Cent-Zeichen                            | &cent;       | &#162;          |
| £       | Pfund-Zeichen                           | &pound;      | &#163;          |
| ¤       | Währungs-Zeichen                        | &curren;     | &#164;          |
| ¥       | Yen-Zeichen                             | &yen;        | &#165;          |
| ¦       | durchbrochener Strich                   | &brvbar;     | &#166;          |
| §       | Paragraph-Zeichen                       | &sect;       | &#167;          |
| ¨       | Pünktchen oben                          | &uml;        | &#168;          |
| ©       | Copyright-Zeichen                       | &copy;       | &#169;          |
| ª       | Ordinal-Zeichen weiblich                | &ordf;       | &#170;          |
| «       | angewinkelte Anführungszeichen links    | &laquo;      | &#171;          |
|         | Verneinungs-Zeichen (nicht darstellbar) | &not;        | &#172;          |
|         | kurzer Trennstrich (nicht darstellbar)  | &shy;        | &#173;          |
| ®       | Registrierte Marke                      | &reg;        | &#174;          |
| ˆ       | Überstrich                              | &macr;       | &#175;          |
| °       | Grad-Zeichen                            | &deg;        | &#176;          |
| ±       | Plusminus-Zeichen                       | &plusmn;     | &#177;          |
| ²       | Hoch-2-Zeichen                          | &sup2;       | &#178;          |
| ³       | Hoch-3-Zeichen                          | &sup3;       | &#179;          |
| ´       | Acute-Zeichen                           | &acute;      | &#180;          |
| µ       | Mikro-Zeichen                           | &micro;      | &#181;          |
| ¶       | Absatz-Zeichen                          | &para;       | &#182;          |
| ·       | Mittelpunkt                             | &middot;     | &#183;          |
| ¸       | Häkchen unten                           | &cedil;      | &#184;          |
| ¹       | Hoch-1-Zeichen                          | &sup1;       | &#185;          |

| Zeichen | Beschreibung                          | Name in HTML | Unicode in HTML |
|---------|---------------------------------------|--------------|-----------------|
| °       | Ordinal-Zeichen männlich              | &ordm;       | &#186;          |
| »       | angewinkelte Anführungszeichen rechts | &raquo;      | &#187;          |
| ¼       | ein Viertel                           | &frac14;     | &#188;          |
| ½       | ein Halb                              | &frac12;     | &#189;          |
| ¾       | drei Viertel                          | &frac34;     | &#190;          |
| ¿       | umgekehrtes Fragezeichen              | &iquest;     | &#191;          |
| À       | A mit Accent grave                    | &Agrave;     | &#192;          |
| Á       | A mit Accent acute                    | &Aacute;     | &#193;          |
| Â       | A mit Circumflex                      | &Acirc;      | &#194;          |
| Ã       | A mit Tilde                           | &Atilde;     | &#195;          |
| Ä       | A Umlaut                              | &Auml;       | &#196;          |
| Å       | A mit Ring                            | &Aring;      | &#197;          |
| Æ       | A mit legiertem E                     | &AElig;      | &#198;          |
| Ç       | C mit Häkchen                         | &Ccedil;     | &#199;          |
| È       | E mit Accent grave                    | &Egrave;     | &#200;          |
| É       | E mit Accent acute                    | &Eacute;     | &#201;          |
| Ê       | E mit Circumflex                      | &Ecirc;      | &#202;          |
| Ë       | E Umlaut                              | &Euml;       | &#203;          |
| Ì       | I mit Accent grave                    | &Igrave;     | &#204;          |
| Í       | I mit Accent acute                    | &Iacute;     | &#205;          |
| Î       | I mit Circumflex                      | &Icirc;      | &#206;          |
| Ï       | I Umlaut                              | &Iuml;       | &#207;          |
| Ð       | Eth (isländisch)                      | &ETH;        | &#208;          |
| Ñ       | N mit Tilde                           | &Ntilde;     | &#209;          |
| Ò       | O mit Accent grave                    | &Ograve;     | &#210;          |
| Ó       | O mit Accent acute                    | &Oacute;     | &#211;          |
| Ô       | O mit Circumflex                      | &Ocirc;      | &#212;          |
| Õ       | O mit Tilde                           | &Otilde;     | &#213;          |
| Ö       | O Umlaut                              | &Ouml;       | &#214;          |
| ×       | Mal- Zeichen                          | &times;      | &#215;          |
| Ø       | O mit Schrägstrich                    | &Oslash;     | &#216;          |
| Ù       | U mit Accent grave                    | &Ugrave;     | &#217;          |
| Ú       | U mit Accent acute                    | &Uacute;     | &#218;          |
| Û       | U mit Circumflex                      | &Ucirc;      | &#219;          |
| Ü       | U Umlaut                              | &Uuml;       | &#220;          |
| Ý       | Y mit Accent acute                    | &Yacute;     | &#221;          |
| Þ       | THORN (isländisch)                    | &THORN;      | &#222;          |
| ß       | scharfes S                            | &szlig;      | &#223;          |
| à       | a mit Accent grave                    | &agrave;     | &#224;          |
| á       | a mit Accent acute                    | &aacute;     | &#225;          |
| â       | a mit Circumflex                      | &acirc;      | &#226;          |
| ã       | a mit Tilde                           | &atilde;     | &#227;          |
| ä       | a Umlaut                              | &auml;       | &#228;          |
| å       | a mit Ring                            | &aring;      | &#229;          |

| Zeichen | Beschreibung       | Name in HTML | Unicode in HTML |
|---------|--------------------|--------------|-----------------|
| æ       | a mit legiertem e  | &aelig;      | &#230;          |
| ç       | c mit Häkchen      | &ccedil;     | &#231;          |
| è       | e mit Accent grave | &egrave;     | &#232;          |
| é       | e mit Accent acute | &eacute;     | &#233;          |
| ê       | e mit Circumflex   | &ecirc;      | &#234;          |
| ë       | e Umlaut           | &euml;       | &#235;          |
| ì       | i mit Accent grave | &igrave;     | &#236;          |
| í       | i mit Accent acute | &iacute;     | &#237;          |
| î       | i mit Circumflex   | &icirc;      | &#238;          |
| ï       | i Umlaut           | &iuml;       | &#239;          |
| ð       | eth (isländisch)   | &eth;        | &#240;          |
| ñ       | n mit Tilde        | &ntilde;     | &#241;          |
| ò       | o mit Accent grave | &ograve;     | &#242;          |
| ó       | o mit Accent acute | &oacute;     | &#243;          |
| ô       | o mit Circumflex   | &ocirc;      | &#244;          |
| õ       | o mit Tilde        | &otilde;     | &#245;          |
| ö       | o Umlaut           | &ouml;       | &#246;          |
| ÷       | Divisions-Zeichen  | &divide;     | &#247;          |
| ø       | o mit Schrägstrich | &oslash;     | &#248;          |
| ù       | u mit Accent grave | &ugrave;     | &#249;          |
| ú       | u mit Accent acute | &uacute;     | &#250;          |
| û       | u mit Circumflex   | &ucirc;      | &#251;          |
| ü       | u Umlaut           | &uuml;       | &#252;          |
| ý       | y mit Accent acute | &yacute;     | &#253;          |
| þ       | thorn (isländisch) | &thorn;      | &#254;          |
| ÿ       | y Umlaut           | &yuml;       | &#255;          |

## D.4 Benannte Zeichen für griechische Buchstaben

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

| Beschreibung | Name in HTML | Unicode in HTML |
|--------------|--------------|-----------------|
| Alpha groß   | &Alpha;      | &#913;          |
| alpha klein  | &alpha;      | &#945;          |
| Beta groß    | &Beta;       | &#914;          |
| beta klein   | &beta;       | &#946;          |
| Gamma groß   | &Gamma;      | &#915;          |
| gamma klein  | &gamma;      | &#947;          |
| Delta groß   | &Delta;      | &#916;          |
| delta klein  | &delta;      | &#948;          |



| Beschreibung                          | Name in HTML | Unicode in HTML |
|---------------------------------------|--------------|-----------------|
| Epsilon groß                          | &Epsilon;    | &#917;          |
| epsilon klein                         | &epsilon;    | &#949;          |
| Zeta groß                             | &Zeta;       | &#918;          |
| zeta klein                            | &zeta;       | &#950;          |
| Eta groß                              | &Eta;        | &#919;          |
| eta klein                             | &eta;        | &#951;          |
| Theta groß                            | &Theta;      | &#920;          |
| theta klein                           | &theta;      | &#952;          |
| Iota groß                             | &Iota;       | &#921;          |
| iota klein                            | &iota;       | &#953;          |
| Kappa groß                            | &Kappa;      | &#922;          |
| kappa klein                           | &kappa;      | &#954;          |
| Lambda groß                           | &Lambda;     | &#923;          |
| lambda klein                          | &lambda;     | &#955;          |
| Mu groß                               | &Mu;         | &#924;          |
| mu klein                              | &mu;         | &#956;          |
| Nu groß                               | &Nu;         | &#925;          |
| nu klein                              | &nu;         | &#957;          |
| Xi groß                               | &Xi;         | &#926;          |
| xi klein                              | &xi;         | &#958;          |
| Omicron groß                          | &Omicron;    | &#927;          |
| omicron klein                         | &omicron;    | &#959;          |
| Pi groß                               | &Pi;         | &#928;          |
| pi klein                              | &pi;         | &#960;          |
| Rho groß                              | &Rho;        | &#929;          |
| rho klein                             | &rho;        | &#961;          |
| Sigma groß                            | &Sigma;      | &#931;          |
| sigmaf klein                          | &sigmaf;     | &#962;          |
| sigma klein                           | &sigma;      | &#963;          |
| Tau groß                              | &Tau;        | &#932;          |
| tau klein                             | &tau;        | &#964;          |
| Upsilon groß                          | &Upsilon;    | &#933;          |
| upsilon klein                         | &upsilon;    | &#965;          |
| Phi groß                              | &Phi;        | &#934;          |
| phi klein                             | &phi;        | &#966;          |
| Chi groß                              | &Chi;        | &#935;          |
| chi klein                             | &chi;        | &#967;          |
| Psi groß                              | &Psi;        | &#936;          |
| psi klein                             | &psi;        | &#968;          |
| Omega groß                            | &Omega;      | &#937;          |
| omega klein                           | &omega;      | &#969;          |
| theta Symbol (nicht darstellbar)      | &thetasym;   | &#977;          |
| upsilon mit Haken (nicht darstellbar) | &upsih;      | &#978;          |
| pi Symbol (nicht darstellbar)         | &piv;        | &#982;          |

## D.5 Benannte Zeichen für mathematische Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

| Beschreibung                                       | Name in HTML | Unicode in HTML |
|----------------------------------------------------|--------------|-----------------|
| für alle (nicht darstellbar)                       | &forall;     | &#8704;         |
| teilweise                                          | &part;       | &#8706;         |
| existiert (nicht darstellbar)                      | &exist;      | &#8707;         |
| leer (nicht darstellbar)                           | &empty;      | &#8709;         |
| nabla (nicht darstellbar)                          | &nabla;      | &#8711;         |
| Element von (nicht darstellbar)                    | &isin;       | &#8712;         |
| kein Element von (nicht darstellbar)               | &notin;      | &#8713;         |
| enthält als Element (nicht darstellbar)            | &ni;         | &#8715;         |
| Produkt                                            | &prod;       | &#8719;         |
| Summe                                              | &sum;        | &#8721;         |
| minus                                              | &minus;      | &#8722;         |
| Asterisk (nicht darstellbar)                       | &lowast;     | &#8727;         |
| Quadratwurzel                                      | &radic;      | &#8730;         |
| proportional zu (nicht darstellbar)                | &prop;       | &#8733;         |
| unendlich                                          | &infin;      | &#8734;         |
| Winkel (nicht darstellbar)                         | &ang;        | &#8736;         |
| und (nicht darstellbar)                            | &and;        | &#8869;         |
| oder (nicht darstellbar)                           | &or;         | &#8870;         |
| Schnittpunkt                                       | &cap;        | &#8745;         |
| Einheit (nicht darstellbar)                        | &cup;        | &#8746;         |
| Integral                                           | &int;        | &#8747;         |
| deshalb (nicht darstellbar)                        | &there4;     | &#8756;         |
| ähnlich wie (nicht darstellbar)                    | &sim;        | &#8764;         |
| annähernd gleich (nicht darstellbar)               | &cong;       | &#8773;         |
| beinahe gleich                                     | &asymp;      | &#8776;         |
| ungleich                                           | &ne;         | &#8800;         |
| identisch mit                                      | &equiv;      | &#8801;         |
| kleiner gleich                                     | &le;         | &#8804;         |
| größer gleich                                      | &ge;         | &#8805;         |
| Untermenge von (nicht darstellbar)                 | &sub;        | &#8834;         |
| Obermenge von (nicht darstellbar)                  | &sup;        | &#8835;         |
| keine Untermenge von (nicht darstellbar)           | &nsup;       | &#8836;         |
| Untermenge von oder gleich mit (nicht darstellbar) | &sube;       | &#8838;         |
| Obermenge von oder gleich mit (nicht darstellbar)  | &supe;       | &#8839;         |
| Direktsumme (nicht darstellbar)                    | &oplus;      | &#8853;         |
| Vektorprodukt (nicht darstellbar)                  | &otimes;     | &#8855;         |
| senkrecht zu (nicht darstellbar)                   | &perp;       | &#8869;         |

| Beschreibung                       | Name in HTML | Unicode in HTML |
|------------------------------------|--------------|-----------------|
| Punkt-Operator (nicht darstellbar) | &sdot;       | &#8901;         |
| Raute                              | &lz;         | &#9674;         |

## D.6 Benannte Zeichen für technische Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

| Beschreibung                              | Name in HTML | Unicode in HTML |
|-------------------------------------------|--------------|-----------------|
| links oben (nicht darstellbar)            | &lceil;      | &#8968;         |
| rechts oben (nicht darstellbar)           | &rceil;      | &#8969;         |
| links unten (nicht darstellbar)           | &lfloor;     | &#8970;         |
| rechts unten (nicht darstellbar)          | &rfloor;     | &#8971;         |
| spitze Klammer links (nicht darstellbar)  | &lang;       | &#9001;         |
| spitze Klammer rechts (nicht darstellbar) | &rang;       | &#9002;         |

## D.7 Benannte Zeichen für Pfeil-Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

| Beschreibung                                 | Name in HTML | Unicode in HTML |
|----------------------------------------------|--------------|-----------------|
| Pfeil links                                  | &larr;       | &#8592;         |
| Pfeil oben                                   | &uarr;       | &#8593;         |
| Pfeil rechts                                 | &rarr;       | &#8594;         |
| Pfeil unten                                  | &darr;       | &#8595;         |
| Pfeil links/rechts                           | &harr;       | &#8596;         |
| Pfeil unten-Knick-links (nicht darstellbar)  | &crarr;      | &#8629;         |
| Doppelpfeil links (nicht darstellbar)        | &lArr;       | &#8656;         |
| Doppelpfeil oben (nicht darstellbar)         | &uArr;       | &#8657;         |
| Doppelpfeil rechts (nicht darstellbar)       | &rArr;       | &#8658;         |
| Doppelpfeil unten (nicht darstellbar)        | &dArr;       | &#8659;         |
| Doppelpfeil links/rechts (nicht darstellbar) | &hArr;       | &#8660;         |

## D.8 Benannte Zeichen für diverse Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

| Beschreibung                               | Name in HTML | Unicode in HTML |
|--------------------------------------------|--------------|-----------------|
| Bullet-Zeichen                             | &bull;       | &#8226;         |
| horizontale Ellipse (nicht darstellbar)    | &hellip;     | &#8230;         |
| Minutenzeichen                             | &prime;      | &#8242;         |
| Überstrich                                 | &oline;      | &#8254;         |
| Bruchstrich                                | &frasl;      | &#8260;         |
| Weierstrass p (nicht darstellbar)          | &weierp;     | &#8472;         |
| Zeichen für "imaginär" (nicht darstellbar) | &image;      | &#8465;         |
| Zeichen für "real" (nicht darstellbar)     | &real;       | &#8476;         |
| Trademark-Zeichen                          | &trade;      | &#8482;         |
| Euro-Zeichen                               | &euro;       | &#8364;         |
| Alef-Symbol (nicht darstellbar)            | &alefsym;    | &#8501;         |
| Pik-Zeichen                                | &spades;     | &#9824;         |
| Kreuz-Zeichen                              | &clubs;      | &#9827;         |
| Herz-Zeichen                               | &hearts;     | &#9829;         |
| Karo-Zeichen                               | &diams;      | &#9830;         |

## D.9 Benannte Zeichen für Interpunktion

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

| Zeichen | Beschreibung                                    | Name in HTML | Unicode in HTML |
|---------|-------------------------------------------------|--------------|-----------------|
|         | Leerzeichen Breite n (nicht darstellbar)        | &ensp;       | &#8194;         |
|         | Leerzeichen Breite m (nicht darstellbar)        | &emsp;       | &#8195;         |
|         | schmales Leerzeichen (nicht darstellbar)        | &thinsp;     | &#8201;         |
|         | null breiter Nichtverbinder (nicht darstellbar) | &zwnj;       | &#8204;         |
|         | null breiter Verbinder (nicht darstellbar)      | &zwj;        | &#8205;         |
|         | Links-nach-rechts-Zeichen (nicht darstellbar)   | &lrm;        | &#8206;         |
|         | Rechts-nach-links-Zeichen (nicht darstellbar)   | &rlm;        | &#8207;         |
| —       | Gedankenstrich Breite n                         | &ndash;      | &#8211;         |
| —       | Gedankenstrich Breite m                         | &mdash;      | &#8212;         |
| ‘       | einfaches Anführungszeichen links               | &lquo;       | &#8216;         |
| ’       | einfaches Anführungszeichen rechts              | &rquo;       | &#8217;         |
| ‚       | einfaches low-9-Zeichen                         | &sbquo;      | &#8218;         |
| “       | doppeltes Anführungszeichen links               | &lquo;       | &#8220;         |
| ”       | doppeltes Anführungszeichen rechts              | &rquo;       | &#8221;         |

| Zeichen | Beschreibung                               | Name in HTML | Unicode in HTML |
|---------|--------------------------------------------|--------------|-----------------|
| „       | doppeltes low-9-Zeichen rechts             | &bdquo;      | &#8222;         |
| †       | Kreuz                                      | &dagger;     | &#8224;         |
| ‡       | Doppelkreuz                                | &Dagger;     | &#8225;         |
| ‰       | zu tausend                                 | &permil;     | &#8240;         |
| ‹       | angewinkeltes einzelnes Anf.zeichen links  | &lquo;       | &#8249;         |
| ›       | angewinkeltes einzelnes Anf.zeichen rechts | &rquo;       | &#8250;         |

## E HTML: Varianten

### E.1 Varianten »strict«, »transitional« und »frameset«

Damit eine HTML-Datei **gültig** ist, muss sie vollständig den Regeln entsprechen, die in der HTML-DTD (*DTD = document type definition*) festgelegt sind. Diese Regeln legen fest, welche Elemente verwendet werden dürfen, wie diese Elemente verschachtelt sein dürfen, welche Attribute zu welchen Elementen erlaubt sind und welche Wertzuweisungen an diese Attribute erlaubt sind. Nun gibt es in HTML 4.0 und auch in HTML 4.01 gleich drei HTML-DTDs. Es handelt sich um verschiedene Varianten.

#### E.1.1 Variante »strict«

Die Variante »strict« ist die vom W3-Konsortium empfohlene und präferierte Variante. Sie zeichnet sich dadurch aus, dass sie nur sehr schlankes HTML erlaubt. Viele Elemente und Attribute zur Formatierung und Visualisierung von Texten fehlen in dieser Variante. Stattdessen sollen Web-Seiten-Autoren CSS Stylesheets zur Formatierung verwenden.

Eine weitere Besonderheit der »strict«-Variante ist, dass innerhalb von `<body>` und `</body>` alle Inhalte in Block-Elementen stehen müssen. Die Variante »strict« eignet sich vor allem für neu zu erstellende Web-Seiten, bei denen zur Formatierung konsequent auf CSS gesetzt wird. Einzelheiten finden Sie unter Anhang E.2 in der Übersicht der nicht erlaubten Elemente und Attribute bei Variante »strict«.

Um die Variante »strict« mit HTML 4.01 explizit zu verwenden, ist ganz zu Beginn einer HTML-Datei folgende Notation erforderlich:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

#### E.1.2 Variante »transitional«

Die Variante »transitional« ist ein Kompromiss, den das W3-Konsortium angesichts eigener Fehler der Vergangenheit (Definitionen aus der HTML-Version 3.2) und angesichts der hohen Verbreitung von Web-Seiten, die jene Features nutzen, anbietet. Die Variante »transitional« erlaubt die Verwendung diverser Elemente und Attribute, die als **deprecated** (*missbilligt*) gekennzeichnet sind und die in der »strict«-Variante nicht mehr vorkommen. Darunter fallen so beliebte HTML-Elemente wie `font` oder häufig verwendete Attribute wie `align=` oder `bgcolor=`.

Bei der »transitional«-Variante ist es auch erlaubt, direkt innerhalb von `<body>` und `</body>` einfach nur Text oder Inline-Elemente zu notieren. Die Variante »transitional« ist daher vor

allen für bereits bestehende Web-Seiten zu empfehlen, die ohne allzu großen Änderungsaufwand auf einen Stand gebracht werden sollen, der gültiges HTML 4.0 oder 4.01 darstellt.

Um die Variante »transitional« in HTML 4.01 explizit zu verwenden, ist ganz zu Beginn einer HTML-Datei folgende Notation erforderlich:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

**Hinweis:** Da sich in dieser Variante der Name der Variante und der Name der DTD unterscheiden, ist mitunter auch von einer »loose«-Variante die Rede. Damit ist die »transitional«-Variante gemeint.

### E.1.3 Variante »frameset«

Die Variante »frameset« ist in HTML 4.01 identisch mit der Variante »transitional«, mit dem einzigen Unterschied, dass die erforderliche Elementverschachtelung unterhalb des Wurzelements `html` eine andere ist. Anstelle eines `body`-Elements ist in dieser Variante ein `frameset`-Element erforderlich.

Um die Variante »frameset« explizit zu verwenden, ist ganz zu Beginn einer HTML-Datei folgende Notation erforderlich:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

## E.2 Nicht erlaubte Elemente und Attribute bei Variante »strict«

Die folgende Tabelle listet auf, welche Elemente und Attribute in der »strict«-Variante **nicht** vorkommen dürfen, weil sie entweder als **deprecated** (*missbilligt*) gekennzeichnet sind oder weil sie zum Thema »Frames« gehören. Nur in der »transitional«-Variante dürfen Elemente und Attribute verwendet werden, die **deprecated** sind. Frame-Definitionen dürfen nur in der »frameset«-Variante verwendet werden.

| Element               | Attribut           | Bemerkung                                                                                                                                                                                                                                                    |
|-----------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>applet</code>   |                    | Element ist deprecated. Java-Applets sollen stattdessen mit dem <code>object</code> -Element eingebunden werden.                                                                                                                                             |
| <code>basefont</code> |                    | Element ist deprecated. Die Standardschriftgröße kann stattdessen mit CSS Stylesheets definiert werden. Mehrere aufeinander folgende HTML-Elemente können in ein <code>div</code> -Element eingeschlossen werden, dem sich CSS-Definitionen zuweisen lassen. |
| <code>body</code>     | <code>alink</code> | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.                                                                                                                                                                    |

| Element  | Attribut   | Bemerkung                                                                                                                                        |
|----------|------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| body     | background | Attribut ist deprecated. Die Hintergrundfarbe kann stattdessen mit CSS Stylesheets definiert werden.                                             |
| body     | bgcolor    | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.                                                        |
| body     | link       | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.                                                        |
| body     | text       | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.                                                        |
| body     | vlink      | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.                                                        |
| br       | clear      | Attribut ist deprecated. Die Angabe zur Textflussfortsetzung kann stattdessen mit CSS Stylesheets definiert werden.                              |
| caption  | align      | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                                                  |
| center   |            | Element ist deprecated. Als Alternative wird das <code>div</code> -Element empfohlen. Die Ausrichtung kann mit CSS Stylesheets definiert werden. |
| div      |            | Element ist deprecated. Als Alternative wird das <code>ul</code> -Element empfohlen.                                                             |
| div      | align      | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                                                  |
| dl       | compact    | Attribut ist deprecated. Die Schriftweite kann stattdessen mit CSS Stylesheets definiert werden.                                                 |
| font     |            | Element ist deprecated. Schriftgröße, Schriftfarbe und Schriftart können stattdessen mit CSS Stylesheets definiert werden.                       |
| frame    |            | Element gehört zum Themenbereich »Frames«.                                                                                                       |
| frameset |            | Element gehört zum Themenbereich »Frames«.                                                                                                       |
| h1-h6    | align      | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                                                  |
| hr       | align      | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                                                  |
| hr       | size       | Attribut ist deprecated. Die Höhe kann stattdessen mit CSS Stylesheets definiert werden.                                                         |
| hr       | noshade    | Attribut ist deprecated. Die Gestaltung kann stattdessen mit CSS Stylesheets definiert werden.                                                   |
| hr       | width      | Attribut ist deprecated. Die Breite kann stattdessen mit CSS Stylesheets definiert werden.                                                       |
| html     | version    | Attribut ist deprecated. Die HTML-Versionsangabe geht aus einer DTD-Deklaration ( <code>&lt;!DOCTYPE . . .&gt;</code> ) hervor.                  |
| iframe   |            | Element kommt in der »strict«-Variante nicht vor.                                                                                                |
| img      | align      | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                                                  |
| img      | hspace     | Attribut ist deprecated. Der Abstand kann stattdessen mit CSS Stylesheets definiert werden.                                                      |



| <i>Element</i> | <i>Attribut</i> | <i>Bemerkung</i>                                                                                                             |
|----------------|-----------------|------------------------------------------------------------------------------------------------------------------------------|
| img            | vspace          | Attribut ist deprecated. Der Abstand kann stattdessen mit CSS Stylesheets definiert werden.                                  |
| img            | border          | Attribut ist deprecated. Der Rahmen kann stattdessen mit CSS Stylesheets definiert werden.                                   |
| input          | align           | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                              |
| isindex        |                 | Element ist deprecated. Als Alternative wird das input-Element mit dem Attribut type="text" empfohlen.                       |
| legend         | align           | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                              |
| li             | type            | Attribut ist deprecated. Der Listentyp kann stattdessen mit CSS Stylesheets definiert werden.                                |
| li             | value           | Attribut ist deprecated. Die Sequenznummer in einer nummerierten Liste kann stattdessen mit CSS Stylesheets geändert werden. |
| menu           |                 | Element ist deprecated. Als Alternative wird das ul-Element empfohlen.                                                       |
| noframes       |                 | Element kommt in der »strict«-Variante nicht vor.                                                                            |
| object         | align           | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                              |
| object         | border          | Attribut ist deprecated. Der Rahmen kann stattdessen mit CSS Stylesheets definiert werden.                                   |
| object         | hspace          | Attribut ist deprecated. Der Abstand kann stattdessen mit CSS Stylesheets definiert werden.                                  |
| object         | vspace          | Attribut ist deprecated. Der Abstand kann stattdessen mit CSS Stylesheets definiert werden.                                  |
| ol             | compact         | Attribut ist deprecated. Die Schriftweite kann stattdessen mit CSS Stylesheets definiert werden.                             |
| ol             | start           | Attribut ist deprecated. Die Startnummer kann stattdessen mit CSS Stylesheets definiert werden.                              |
| ol             | type            | Attribut ist deprecated. Der Listentyp kann stattdessen mit CSS Stylesheets definiert werden.                                |
| p              | align           | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                              |
| pre            | width           | Attribut ist deprecated. Die Elementbreite kann stattdessen mit CSS Stylesheets definiert werden.                            |
| s              |                 | Element ist deprecated. Durchgestrichener Text kann stattdessen mit CSS Stylesheets definiert werden.                        |
| script         | language        | Attribut ist deprecated. Stattdessen wird das Attribut type= empfohlen.                                                      |
| strike         |                 | Element ist deprecated. Durchgestrichener Text kann stattdessen mit CSS Stylesheets definiert werden.                        |
| table          | align           | Attribut ist deprecated. Die Ausrichtung kann stattdessen mit CSS Stylesheets definiert werden.                              |

| <i>Element</i> | <i>Attribut</i> | <i>Bemerkung</i>                                                                                    |
|----------------|-----------------|-----------------------------------------------------------------------------------------------------|
| table          | bgcolor         | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.           |
| td             | bgcolor         | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.           |
| td             | height          | Attribut ist deprecated. Die Elementhöhe kann stattdessen mit CSS Stylesheets definiert werden.     |
| td             | nowrap          | Attribut ist deprecated. Der Umbruch kann stattdessen mit CSS Stylesheets kontrolliert werden.      |
| td             | width           | Attribut ist deprecated. Die Elementbreite kann stattdessen mit CSS Stylesheets definiert werden.   |
| th             | bgcolor         | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.           |
| th             | height          | Attribut ist deprecated. Die Elementhöhe kann stattdessen mit CSS Stylesheets definiert werden.     |
| th             | nowrap          | Attribut ist deprecated. Der Umbruch kann stattdessen mit CSS Stylesheets kontrolliert werden.      |
| th             | width           | Attribut ist deprecated. Die Elementbreite kann stattdessen mit CSS Stylesheets definiert werden.   |
| tr             | bgcolor         | Attribut ist deprecated. Die Farbe kann stattdessen mit CSS Stylesheets definiert werden.           |
| u              |                 | Element ist deprecated. Unterstrichener Text kann stattdessen mit CSS Stylesheets definiert werden. |
| ul             | compact         | Attribut ist deprecated. Die Schriftweite kann stattdessen mit CSS Stylesheets definiert werden.    |
| ul             | type            | Attribut ist deprecated. Der Listentyp kann stattdessen mit CSS Stylesheets definiert werden.       |

# Stichwortverzeichnis

XE 380  
 # 197, 423  
 #CDATA 1225  
 \$[1..9] 957  
 % 204, 427  
 & 204, 395  
 &#160; 149  
 &nbsp; 149  
 \* 274, 417  
 .asp 112  
 .class 345, 358  
 .css 407, 409  
 .csv 1035  
 .eot 445, 1107  
 .gif 1111  
 .htaccess 1195  
 .htgroups 1198  
 .htm 112, 389  
 .html 112, 389  
 .htusers 1197  
 .java 825  
 .jpe 1112  
 .jpeg 1112  
 .jpg 1112  
 .lwf 1114  
 .ocx 348  
 .pfr 445, 1107  
 .php 112  
 .png 1113  
 .profile 1195  
 .shtm 112  
 .shtml 112  
 .svg 1115  
 .svgz 1115  
 .swf 348  
 .xhtml 389  
 .zip 359  
 // 109  
 :active 489  
 :after 491  
 :before 491  
 :first-child 490  
 :first-letter 490  
 :first-line 490  
 :focus 489  
 :hover 489  
 :link 489  
 :visited 489  
 @ 36, 200  
 @font-face 444  
 @import 410  
 @media 411  
 @page 511  
 \_blank 131, 194, 289  
 \_parent 131, 194, 289, 300  
 \_self 194, 288, 300  
 \_top 131, 194, 300  
 < 395  
 <!-- 107  
 <!-- ... --> 407  
 <![CDATA[...]]> 396  
 <?xml?> 389  
     encoding 389  
     version 389  
 <a> 191, 394, 702  
     accesskey 212  
     charset 210  
     href 191  
     hreflang 210  
     id 394  
     name 197, 394  
     rel 208  
     rev 208  
     tabindex 211

- target 194, 278, 288
- type 208
- <abbr> 168, 703
- <acronym> 168, 704
- <address> 161, 704
- <applet> 358, 705
  - align 361
  - alt 359
  - archive 359
  - code 358
  - codebase 358
  - height 359
  - mayscript 359
  - name 359
  - object 359
  - width 359
- <area> 230, 706
  - accesskey 231
  - alt 231
  - coords 230
  - href 231
  - nohref 231
  - shape 230
  - tabindex 231
  - title 231
- <b> 173, 707
- <base target=> 131
- <base> 707
  - href 130
  - target 131
- <basefont> 183, 708
- <bdo> 170, 709
  - dir 170
- <bgsound> 140
  - loop 140
  - src 140
- <big> 173, 709
- <blink> 188
- <blockquote> 159, 710
  - cite 160
  - type 160
- <body> 94, 711
  - alink 136
  - background 137
  - bgcolor 135
  - bgproperties 138
  - leftmargin 138
  - link 136
  - marginheight 139
  - marginwidth 139
  - text 136
  - topmargin 138
  - vlink 136
- <br> 148, 355, 712
  - clear 223, 260, 355, 361
- <button> 321, 713
  - accesskey 329
  - disabled 331
  - tabindex 328
  - type 321
  - type= 334
  - value 322
- <caption> 258, 714
  - align 258
- <center> 175, 715
- <cite> 168, 715
- <code> 168, 715
- <col> 237, 716
  - span 238
  - width 237
- <colgroup> 237, 717
  - span 237
  - width 237
- <dd> 154, 718
- <del> 169, 719
  - datetime 169
- <dfn> 168, 720
- <dir> 155, 720
- <div> 175, 420, 721
  - align 175
- <dl> 154, 722
  - compact 157
- <dt> 154, 722
- <em> 168, 723
- <embed> 140, 362
  - align 363
  - autostart 140, 363
  - height 140, 362
  - hidden 140, 363
  - hspace 363

- loop 140, 363
- palette 363
- pluginspage 363
- pluginurl 362
- src 140, 362
- vspace 363
- width 140, 362
- <fieldset> 326, 724
- <font> 182, 725
  - color 182
  - face 182
  - size 182
- <form> 298, 726
  - accept-charset 300
  - action 299
  - enctype 299, 324
  - method 299
  - target 300
- <frame> 727
  - frameborder 286
  - longdesc 281
  - marginwidth 283
  - name 278, 288, 678
  - noresize 284
  - scrolling 282
  - src 278
- <frameset> 273, 728
  - border 286
  - bordercolor 287
  - cols 274
  - frameborder 286
  - framespacing 286
  - rows 273
- <h?> 142
  - align 142
- <h1> 729
- <head> 730
  - profile 129
- <hr> 178, 730
  - align 179
  - color 180
  - noshade 179
  - size 179
  - width 179
- <html> 94, 731
  - xmlns 391
- <i> 173, 732
- <iframe> 290, 732
  - align 292
  - height 291
  - hspace 292
  - longdesc 292
  - name 290
  - src 290
  - vspace 292
  - width 291
- <ilayer> 368
  - above 374
  - background 372
  - below 374
  - bgcolor 372
  - id 368
  - left 368
  - name 368
  - top 368
  - visibility 376
  - z-index 375
- <img> 214, 733
  - align 221, 222
  - alt 215
  - border 218
  - height 216
  - hspace 222
  - ismap 232
  - longdesc 219
  - lowsrc 215
  - name 218
  - src 214
  - usemap 231
  - vspace 222
  - width 216
- <input> 301, 735
  - accept 323
  - accesskey 329
  - checked 318, 319
  - disabled 331
  - maxlength 323
  - maxLength 302

- name 301
- readonly 310
- size 302
- src 334
- tabindex 328
- type= 301, 304, 318, 319, 320, 323, 325, 333, 334
- value 304, 318, 319, 325
- <ins> 169, 737
  - cite 169
  - datetime 169
- <isindex> 135, 737
  - prompt 135
- <kbd> 168, 738
- <label> 327, 738
  - accesskey 329
- <layer> 367, 1012
  - above 374
  - background 372
  - below 374
  - bgcolor 372
  - clip 369
  - height 367
  - id 367
  - left 367
  - name 367
  - pagex 371
  - pagey 371
  - src 369
  - top 367
  - visibility 376
  - width 367
  - z-index 375
- <legend> 326, 739
  - accesskey 329
- <li> 151, 740
  - value 157
- <link> 132, 407, 409, 741, 1108
  - charset 133
  - href 133, 407
  - hreflang 133
  - media 133, 409
  - rel 132, 407
  - rev 134
  - target 133
  - title 133
  - type 407
- <listing> 165
- <map> 229, 742
  - name 229
- <marquee> 185
  - align 185
  - behavior 185
  - bgcolor 185
  - direction 185
  - height 185
  - hspace 185
  - scrollamount 185
  - scrolldelay 185
- <menu> 155, 743
- <meta> 120, 744
  - content 120
  - http-equiv 120
  - lang 125
  - name 120
  - scheme 129
- <multicol> 186
  - cols 186
  - gutter 187
  - width 187
- <nobr> 148
- <noembed> 364
- <noframes> 279, 744
- <noscript> 385, 745
- <object> 341, 342, 746
  - align 352
  - border 351
  - classid 345, 347, 349
  - codebase 346, 348, 350
  - codetype 346
  - data 343
  - declare 356
  - height 343
  - hspace 354
  - id 356
  - standby 356
  - type 343
  - usemap 345
  - vspace 354
  - width 343

- <ol> 153, 747
  - compact 157
  - start 157
  - type 157
- <optgroup> 316, 748
  - disabled 331
  - label 316
- <option> 311, 749
  - disabled 331
  - selected 313
  - value 315
- <p> 145, 750
  - align 145
- <param> 347, 348, 359, 751
  - name 347, 348, 350, 359
  - value 347, 348, 350
- <plaintext> 165
- <pre> 164, 752
  - width 165
- <q> 168, 752
- <s> 173, 753
- <samp> 168, 754
- <script> 383, 754
  - charset 385
  - defer 385
  - language! 383
  - src 385
  - type 383
- <select> 311, 756
  - disabled 331
  - multiple 312
  - name 311
  - size 311
  - tabindex 328
- <small> 173, 757
- <spacer> 187
  - align 188
  - height 188
  - size 188
  - type 187
  - width 188
- <span> 176, 420, 758
- <strike> 173, 759
- <strong> 168, 759
- <style> 381, 406, 760
  - media 382
  - type 382
- <sub> 173, 761
- <sup> 173, 761
- <table> 235, 761
  - align 260
  - background 253
  - bgcolor 252
  - border 235
  - bordercolor 254
  - bordercolordark 254
  - bordercolorlight 254
  - cellpadding 241
  - cellspacing 241
  - datapagesize 1037
  - frame 243
  - height 245
  - hspace 260
  - rules 245
  - summary 266
  - vspace 260
  - width 245
- <tbody> 240, 764
  - align 251
  - background 253
  - char, charoff 251
  - valign 251
- <td> 235, 765
  - abbr 267
  - align 251
  - background 253
  - bgcolor 252
  - char, charoff 251
  - colspan 255
  - headers 268
  - nowrap 249
  - rowspan 256
  - valign 251
- <textarea> 305, 766
  - accesskey 329
  - cols 305
  - disabled 331
  - name 305
  - readonly 310
  - rows 305

- tabindex 328
- wrap 308
- <tfoot> 240, 768
  - align 251
  - background 253
  - char, charoff 251
  - valign 251
- <th> 235, 769
  - abbr 267
  - align 251
  - axis 269
  - background 253
  - bgcolor 252
  - char, charoff 251
  - colspan 255
  - id 267
  - nowrap 249
  - rowspan 256
  - scope 268
  - valign 251
- <thead> 239, 770
  - align 251
  - background 253
  - char, charoff 251
  - valign 251
- <title> 94, 119, 771
- <tr> 235, 771
  - align 251
  - background 253
  - bgcolor 252
  - char, charoff 251
  - valign 251
- <tt> 173, 773
- <u> 173, 773
- <ul> 151, 774
  - compact 157
  - type 157
- <var> 168, 775
- <wbr> 149
- <xmp> 165
- > 395, 417
- > 107
- border-top-color: 466
- 404-Error 1163

**A**

- a 702
  - active 193
  - hover 193
  - link 193
  - visited 193
- abbr 267, 703
- Abmahnung, Datensparsamkeit 298
- about, blank 669
- above 243, 374, 523, 863
- abs() 933
- Absatzkontrolle 448
- absbottom 221, 353
- absmiddle 221, 353
- absolute 496
- Absolute Pfadangaben 110
- Abstand 454
- accept 323
- accept-charset 300
- access.log 1160
- accesskey 212, 231, 329
- acos() 933
- acronym 704
- action 299, 828
- Active Server Pages 85
- activeborder 429
- activecaption 429
- ActiveX 88
- ActiveX-Controls 347
- AddAlt 1203
- AddAltByType 1203
- AddDescription 1202
- AddEncoding 1202
- addEventListener() 1000, 1083
- AddIcon 1202
- AddIconByEncoding 1202
- AddIconByType 1202
- address 704
- Adressbasis 130
- Adressen 160
  - formatieren mit CSS 161
- Adresszeile 654
- Advanced Research Projects Agency 27
- Aladin 1192



- alert() 660
- align 145, 221, 251, 292, 352, 354, 361, 363
- alignment: 453
- alink 136
- alinkColor 682
- all 223, 245, 361, 796, 1003
  - className 799
  - click() 810
  - contains() 810
  - dataFld 799
  - dataFormatAs 800
  - dataPageSize 800
  - dataSrc 800
  - getAttribute() 811
  - id 801
  - innerHTML 801
  - innerText 802
  - insertAdjacentHTML() 811
  - insertAdjacentText() 812
  - isTextEdit 802
  - lang 803
  - language 803
  - length 803
  - offsetHeight 804
  - offsetLeft 804
  - offsetParent 805
  - offsetTop 806
  - offsetWidth 806
  - outerHTML 806
  - outerText 807
  - parentElement 807
  - parentTextEdit 808
  - recordNumber 809
  - removeAttribute() 813
  - scrollIntoView() 813
  - setAttribute() 814
  - sourceIndex 809
  - tagName 809
  - title 809
- Allow 1194
- Allow from 1199
- allowDelete 1052
- allowInsert 1052
- allowUpdate 1053
- Alphakanal 1129
- alt 215, 231, 359
- altavista 1156
- altKey 885
- always 489, 515
- amaya 1192
- anchor() 964
- anchors 821
  - length 823
  - name 822
  - text 823
  - x 823
  - y 824
- Änderungsmarkierungen 169
- Andreessen, Marc 43
- Anführungszeichen 101
- Animierte GIF-Grafik 1140
- Animiertes GIF, Hintergrundbild 138
- Anker 196
  - Referenzieren 109
  - reservierte Namen 197
- anonymous 38
- ANSI-Zeichensatz 1093
- Anti-Aliasing 1139
- AOL 1151
- Apache 1195
- appName 939
- appendChild() 729, 764, 785, 790, 998
- AppendData 1048, 1050
- appendData() 786
- applet 705
- Applet 87
- applets 824
  - length 826
- application 1179
- application/font-tdpfr 1109
- application/xhtml+xml 388
- application/xml 388
- appName 940
- appVersion 940
- appworkspace 429
- aqua 103
- archive 359
- Arcus Cosinus 933
- Arcus Sinus 933
- Arcus Tangens 934

area 706  
 arguments 928  
 arity 928  
 ARPA-Net 27  
 Array 683, 897  
   concat() 901  
   join() 902  
   length 900  
   pop() 903  
   push() 903  
   reverse() 904  
   shift() 904  
   slice() 905  
   sort() 906  
   splice() 906  
   unshift() 907  
 ascent: 447  
 ASCII-Zeichensatz 110, 1093, 1097  
 asin() 933  
 ASP 85, 112  
 Assoziierter Knoten 987  
 atan() 934  
 Attribut  
   in HTML-Tags 92  
   Universalattribut 93  
 Attribute  
   deprecated 1225  
   implied 1225  
   required 1225  
 attributes 777  
 Attributknoten 987  
 AU 140  
 audio 1179  
 Aufzählungslisten 150  
 Augment 55  
 Ausrichten, am Dezimalzeichen 251  
 Ausrichtung 448  
 Außenrand 454  
 Aussprache 518  
 Auswahlliste  
   Mehrfachauswahl 312  
   verschachtelte Struktur 315  
 Auswahllisten 310, 842  
 AuthGroupFile 1196  
 AuthName 1196

author 121  
 AuthType 1196  
 AuthUserFile 1196  
 auto 485, 497, 498, 499, 501, 503, 510, 512, 515,  
   521  
 Autor 108  
   Meta-Angabe 121  
 Autorenrechte 108  
 autostart 140, 363  
 availHeight 959  
 availWidth 960  
 avoid 515  
 axis 269  
 azimuth: 521

## B

b 707  
 back() 661, 892  
 Backbone 29  
 background 137, 253, 363, 372, 429, 476, 864  
 background-attachment: 475  
 background-color 193, 471  
 background-image: 472  
 background-position: 475  
 background-repeat: 473  
 Backtracking 52  
 base 707  
 basefont 708  
 baseline 221, 251, 353, 448, 451  
 Basic Multilingual Plane 1102  
 Baummetapher 1148  
 Baumstruktur 987  
 bcc 202  
 bdo 709  
 behavior 185  
 below 243, 374, 523, 865  
 Benannte Seiten 516  
 Benannte Zeichen 100  
 Benutzereingabe 671  
 Berechnungen 930  
 Berners-Lee, Tim 41  
 bgcolor 135, 185, 252, 372, 683, 864  
 bgproperties 138  
 big 709  
 big() 965

Bild, siehe Grafik 213  
 Bildbearbeitung 1126  
 Bilddatei, Groß-/Kleinschreibung 215  
 Bilder 850  
 Bildlaufleisten 281  
 Bitstream 1106  
 black 103  
 blendTrans() 1057  
 Blinde Pixel 1124  
 Blinde Tabelle 223  
 blink 442  
 blink() 965  
 Blinkender Text 188  
 block 507  
 Block-Element 167, 174  
 blockquote 116, 710  
 Blocksatz 142, 145  
 blue 103  
 blur() 661, 839  
 BMP 1102  
 Body 93, 203, 711  
 bold 438  
 bold() 965  
 bolder 438  
 Bookmarks 52  
 Boolean 908  
 Boole'sches Objekt 908  
 border 218, 235, 286, 351, 469, 852  
 border-bottom: 471  
 border-bottom-color: 467  
 border-bottom-style: 469  
 border-bottom-width: 466  
 border-collapse: 469, 485  
 border-color: 254, 287, 466  
 bordercolordark 254  
 bordercolorlight 254  
 border-left: 471  
 border-left-color: 466  
 border-left-style: 469  
 border-left-width: 465  
 border-right: 471  
 border-right-color: 467  
 border-right-style: 469  
 border-right-width: 466  
 border-spacing: 487

border-style: 468  
 border-top: 471  
 border-top-style: 469  
 border-top-width: 465  
 border-width: 465  
 both 506  
 bottom 221, 353, 361, 451  
 box 243  
 br 712  
 broken image 215  
 Bubble-Effekt 1010  
 Buchmetapher 1148  
 Bullet 1117  
 Bush, Vannevar 54  
 Button 319, 320, 321, 713, 1116  
 buttonface 429  
 buttonhighlight 429  
 Buttonleiste 660  
 buttontext 429

## C

cabbase 1052  
 Cailliau, Robert 42  
 caller 929  
 Candidate Recommendation 61  
 cap-height: 447  
 capitalize 443  
 caption 714  
 caption-side: 484  
 captiontext 429  
 captureEvents() 661, 690, 873, 1031  
 CaseSensitive 1048, 1051  
 cc 202  
 ceil() 934  
 cellpadding 228, 241  
 cellspacing 228, 241  
 center 145, 453, 522, 715  
 center behind 522  
 center-left 522  
 center-left behind 522  
 centerline: 448  
 center-right 522  
 center-right behind 522  
 CERN 41  
 CGI 83, 339

- cgi-bin 83
- CGI-Script 68, 109
- CGI-Services 68
- char 251
- character data 1225
- charAt() 966
- charCodeAt() 966
- charoff 251
- CharSet 210, 385, 684, 1049, 1051
- Chat 39
- checkbox 318, 319
- checked 318, 319, 835
- child 524
- childNodes 778
- circle 157, 230, 479
- cite 160, 169, 715
- cjk-ideographic 480
- class 419, 421
- classid 345, 347, 349
- className 799
- clear 223, 260, 355, 361, 505
- clearInterval() 662
- clearTimeout() 663
- click() 810, 840
- Client-Identifikation 1189
- Client-Push 32
- Client-Server-Technologie 31
- clientX 886
- clientY 886
- clip 369, 510, 865
- Cliparts 1121
- cloneNode() 786
- close() 652, 664, 691
- closed 652
- CLSID: 347, 349
- cm 427
- code 358, 526, 715
- codebase 346, 348, 350, 358
- col 268, 716
- colgroup 268, 717
- collapse 469, 486, 488
- color 182, 193, 443
- colorDepth 960
- cols 186, 245, 274
- colspan 255
- COM 88
- Common Gateway Interface 83, 339
- compact 157, 507
- compatible 1190
- complete 852
- Component Object Model 88
- CompuServe 1151
- concat() 901, 966
- condensed 438
- confirm() 664, 677
- contains() 810
- content 120, 492
- Content-Script-Type 126
- Content-Style-Type 126
- content-type 125
- continous 526
- cookie 685
- cookieEnabled 686, 940
- Cookies 685, 940
- cookies.txt 685
- Cooperation for an Open Systems  
Interconnection Networking in Europe 29
- Coordinated Universal Time 169
- coords 230
- Corporate Design 1148
- cos() 934, 995
- COSINE 29
- Cosinus 934
- Counter 69, 493
- counter-increment: 494
- counter-reset: 494
- Crawler 1192
- createAttribute() 691, 784
- createElement() 692, 729, 764, 784, 790, 998
- createTextNode() 693, 784, 790, 998
- Credits 108
- crop 513
- cross 513
- crosshair 528
- CSS
  - Attribut-bedingte Formate 417
  - Drucklayout 511
  - Elemente positionieren 494
  - Farbangaben 428
  - Filter 530

- Formate für HTML-Elemente 413
- Formate für Klassen 419
- Individualformate 422
- Kommentare 411
- Leerzeichen beachten 413
- Pseudoformate 424, 489
- Sprachausgabe 517
- Stylesheets 77, 403
- zentrale Formate 380, 412
- CSS-Datei 407
- CSS-Versionen 78
- ctrlKey 885
- cue: 520
- cue-after: 520
- cue-before: 519
- cursive 434
- cursor: 527

## D

- Danksagung 108
- dashed 468
- data 343, 779
- dataFld 799, 1038
- dataFormatAs 800, 1038
- dataPageSize 800
- dataSrc 800
- DataURL 1049
- date 121
- Date 909, 983
  - getDate() 911
  - getDay() 912
  - getFullYear() 912
  - getHours() 913
  - getMilliseconds() 913
  - getMinutes() 914
  - getMonth() 914
  - getSeconds() 915
  - getTime() 915
  - getTimezoneOffset() 916
  - getUTCDate() 916
  - getUTCDay() 916
  - getUTCFullYear() 917
  - getUTCHours() 917
  - getUTCMilliseconds() 917
  - getUTCMinutes() 917

- getUTCMonth() 917
- getUTCSeconds() 917
- getYear() 917
- parse() 918
- setDate() 918
- setFullYear() 919
- setHours() 919
- setMilliseconds() 920
- setMinutes() 920
- setMonth() 921
- setSeconds() 922
- setTime() 922
- setUTCDate() 923
- setUTCDay() 923
- setUTCFullYear() 923
- setUTCHours() 923
- setUTCMilliseconds() 923
- setUTCMinutes() 924
- setUTCMonth() 924
- setUTCSeconds() 924
- setYear() 924
- toGMTString() 925
- toLocaleString() 925
- UTC() 925
- Dateiendungen 112
- Dateiname
  - Dateiendungen 112
  - Default-Dateinamen 112
  - Default-Startseite 200
  - Erlaubte Zeichen 112
  - Groß-/Kleinschreibung 215
  - Kompatibilität 112
  - Konventionen von Unix 111
- Dateiorganisation 113
- Datei-Upload 322
- Datenanbindung 1034
- Datenbank durchsuchen 1038
- Datendatei als Objekt 342
- Datenmodell
  - JDBC DataSource Applet 1051
  - MSHTML Data Source 1054
  - RDS, Remote Data Service 1050
  - TDC, Tabular Data Control 1048
  - XML Data Source 1053
- Datenpaket 30

- Datenschutz 1170
- Datenzelle 235
- datetime 169
- Datum 910
- dbURL 1052
- DC.Contributor 123
- DC.Coverage 124
- DC.Creator 122
- DC.Date 123
- DC.Description 122
- DC.Format 123
- DC.Identifizier 123
- DC.Language 124
- DC.Publisher 123
- DC.Relation 124
- DC.Rights 124
- DC.Source 123
- DC.Subject 122
- DC.Title 122
- DC.Type 123
- DCT 1112
- dd 718
- decimal 478
- decimal-leading-zero 480
- declare 356
- decodeURI() 978, 984
- decodeURIComponent() 978
- default.htm 112
- default.html 112
- defaultCharset 687
- defaultChecked 836
- DefaultIcon 1202
- defaultSelected 845
- Default-Startseite 200
- defaultStatus 651, 652
- defaultValue 837
- defer 385
- Definitionen 413
- Definitionslisten 153
- definition-src, url() 448
- deg 431, 521, 522
- del 719
- deleteData() 787
- DENIC 33
- DE-NIC 35
- Deny from 1199
- deprecated 101, 1225
- descent: 448
- description 121, 944, 947
- Deutsches Forschungsnetz 29
- Deutsches Network Information Center 33, 35
- Dezimalzeichen, ausrichten am 251
- dfn 720
- DFN 29
- DHTML 985
  - Beispiel, Bilderbuch 1087
  - Beispiel, Laufende Uhr 1084
  - Beispiel, Navigationsleiste 1078
  - Datenanbindung bei Microsoft 1034
  - DOM-Modell 986
  - Funktionsbibliothek 1067
  - Microsoft-Modell 1001
  - Netscape-4-Modell 1012
- Dialogfenster 664, 671
- Dicktengleiche Schrift 163
- digits 526
- dino-online 1156
- dir 170, 378, 720
- direction 185
- direction: 504
- DirectoryIndex 1200
- disabled 331
- disableExternalCapture() 665
- Disallow 1194
- disc 157, 479
- Diskrete Cosinus Transformation 1112
- Diskussionsforen 70
- display: 507
- Dithering 1130
- div 721
- dl 722
- DNS 32
- DOCTYPE 96
- document 681, 867
  - alinkColor 682
  - bgColor 683
  - captureEvents() 690
  - charset 684
  - close() 691
  - cookie 685

- createAttribute() 691
- createElement() 692
- createTextNode() 693
- defaultCharset 687
- fgColor 687
- getElementById() 693
- getElementsByName() 694
- getElementsByTagName() 695
- getSelection() 696
- handleEvent() 697
- lastModified 688
- linkColor 688
- open() 697
- referrer 689
- releaseEvents() 698
- routeEvent() 698
- title 689
- URL 689
- vlinkColor 690
- write() 698
- writeln() 699
- Document Object Model 81, 388, 986
- Document Type Definition 387
- Dokumenttyp
  - Frames 272
  - Frameset 97
  - Strict 96
  - Transitional 97
- Dokumenttyp-Angabe 95
- Dokumenttyp-Definitionen 95
- DOM 81, 388, 986
- DOMActivate 1000
- Domain 32, 1150, 1152
- Domain Name Service 32
- Domain-Grabbing 33
- DOMFocusIn 1000
- DOMFocusOut 1000
- DOMNodeInserted 1000
- DOMNodeRemoved 1000
- DOMSubtreeModified 1000
- Dot 1117
- Dots per Inch 1127
- dotted 468
- double 468
- Download 38
- Downloadbare Schriftarten 1105
- Download-Verweise 205
- dpi 1127
- Dropdown-Liste 311
- Drucken 671
- Drucklayout 511
- DSSSL 404
- dt 722
- DTD 95, 387
- Dublin Core 122
- Dynamic HTML 82, 985
- Dynamische Filter 1054
- Dynamisches HTML 797, 985
- E**
- E 930
- EBCDIC-Zeichensatz 1093
- Ebone 29
- ECMA 1096
- Eingabefelder 301
  - Zeilenumbruch 307
- Eingebettete Frames 289
- Elementbaum 691, 692, 700
- Elemente 91
- Elementknoten 987
- elements 833
  - blur() 839
  - checked 835
  - click() 840
  - defaultChecked 836
  - defaultValue 837
  - handleEvent() 841
  - name 838
  - select() 842
  - type 838
  - value 839
- elevation: 522
- em 427, 723
- E-Mail 36
  - Verschlüsselung 37
- E-Mail-Verweis 161, 200
  - Sonderzeichen 204
  - zusätzliche Angaben 202
- embeds 857
  - height 858

- hspace 859
- length 859
- name 859
- play() 861
- src 860
- stop() 861
- type 860
- vspace 860
- width 860
- empty-cells: 487
- enabledPlugin 944
- enableExternalCapture() 665
- encodeURIComponent() 978
- encodeURIComponent() 979
- encoding 389, 829
- enctype 299, 324
- Engelbart, Douglas C. 55
- Enquire 42
- Ereignisüberwachung 883
- e-resize 528
- Erkennungssignal 519
- ErrorDocument 1200
- escape() 979
- EscapeChar 1049
- Eule 1192
- Eulersche Konstante 930
- EuropaNET 29
- European Computer Manufacturer's Association 1096
- Eurozeichen 101
- eval() 980, 994
- event 883
  - altKey 885
  - clientX 886
  - clientY 886
  - ctrlKey 885
  - keyCode 886
  - layerX 887
  - layerY 887
  - modifiers 887
  - offsetX 888
  - offsetY 888
  - pageX 888
  - pageY 888
  - screenX 889

- screenY 889
- shiftKey 885
- type 890
- which 890
- x 891
- y 891
- Event-Handler 232, 379, 674, 690, 883, 1009
  - überwachen 661
  - Vererbung bei Microsoft 1010
- Event-Handling im DOM 998
- ex 427
- exec() 958
- ExecCGI 1200
- exp() 935
- expanded 439
- expires 127
- Extensible HyperText Markup Language 387
- Extensible Markup Language 387
- Extensible Stylesheet Language 80, 388
- Extensions 112
- extra-condensed 439
- extra-expanded 439

## F

- face 182
- false 908
- FancyIndexing 1202
- fantasy 434
- FAQ 63
- Farbangaben 428
- Farbe
  - Hintergrund 135
  - Text 135
  - Verweise 135
- Farbe für Text 687
- Farben
  - Definieren in HTML 101
  - Hexadezimal-Modus 102
  - RGB-Wert 102
- Farbnamen
  - 16 Grundfarben 104
  - Netscape-Farbnamen 104
- Farbpaletten 1111
- Farbtiefe 1128
- far-left 522



- far-left behind 522
- fast 523
- faster 523
- Feedback 108, 1164
- Fehlermeldung (HTTP) 1173
- female 524
- Fenster öffnen 669
- Fenster schließen 664
- Fenster verschieben 667, 668
- Fenstergröße ändern 673
- fgColor 687
- FieldDelim 1049
- fieldset 724
- file 200, 323
- File Transfer Protocol 38, 1154
- filename 948
- filter
  - Alpha() 531
  - Blur() 532
  - Chroma() 534
  - DropShadow() 534
  - FlipH() 535
  - FlipV() 536
  - Glow() 536
  - Gray() 537
  - Invert() 537
  - Mask() 538
  - Shadow() 539
  - Wave() 539
  - XRay() 540
- Filter 530, 1042, 1049, 1054
- find() 665
- finishopacity 532
- finishx 532
- finisly 532
- fireball 1156
- Fireball 1192
- firstChild 696, 703, 780, 990
- Fish-Eye-Views 51
- fixed 475, 485, 496
- fixed() 967
- Flash 89
  - als Objekt einbinden 348
- float 227, 504
- floor() 935
- focus() 666, 841
- font 439, 725
- fontcolor() 967
- font-family: 434, 445
- fontsize() 968
- font-size: 436, 446
- font-stretch: 438
- font-style: 435, 446
- font-variant: 436
- font-weight 193, 438, 446
- foreground 363
- form 726, 837
- Formatierung
  - logisch 167
  - physisch 167, 171
- forms 826
  - action 828
  - encoding 829
  - focus() 841
  - form 837
  - handleEvent() 832
  - length 829
  - method 830
  - name 831
  - reset() 832
  - submit() 832
  - target 831
- Formular
  - Auswahllisten 842
  - Formularbereich definieren 298
- Formulare 297, 827
  - Abbrechen-Button 331
  - Auswahllisten 310
  - Beschriftung 327
  - Button 319
  - Eingabefelder 301
  - Elemente gruppieren 325
  - grafische Buttons 333
  - Label für Elemente 327
  - mehrzeilige Eingabefelder 305
  - Passwort-Eingabefelder 304
  - Sende-Button 331
  - Tabulatorsteuerung 328
  - Tastaturkürzel 329
  - verarbeiten mit CGI 339

- verarbeiten per E-Mail 338
  - Versteckte Elemente 324
  - Zeichensatz 300
  - Zielfenster für Server-Antwort 300
- Formularelemente 833
- Forum 69
- forward() 666, 892
- Forwarding 127
- Fotos 1124
- frame 243, 727
- Frame, Innenabstände 282
- frameborder 286
- frames 677
  - length 680
- Frames 271, 678
  - eingebettete Frames 289
  - Farbige Rahmen 287
  - Frameset definieren 273
  - Langbeschreibung für Framefenster 281
  - Nachteile 294
  - Noframes-Bereich 279
  - Ohne Rahmen 284
  - Rahmendicke 284
  - Unveränderbare Fenstergröße 284
  - Verweise in Frames 287
  - Zielfensterbasis 131
- frameset 728, 1274
- Frameset 97
  - HTML-Grundgerüst 272
  - siehe Frames 272
- framespacing 286
- Frequently Asked Questions 63
- fromCharCode() 968
- ftp 109
  - // 199
- FTP 38, 1154
- FTP 199
- FTP-Programme 67, 1151, 1154
- fuchsia 103
- Function 926
  - arguments 928
  - arity 928
  - caller 929

**G**

- Gammakorrektur 1135
- Gästebuch 68
- Gateway 34
- Gatterzeichen 197
- Geschützte Leerzeichen 149
- get 299
- getAttribute() 788, 811, 819, 1074
- getAttributeNode() 788
- getDate() 911
- getDay() 912
- getElementById() 378, 693, 988
- getElementsByName() 694, 988
- getElementsByTagName() 695, 988
- getFullYear() 912
- getHours() 913
- getMilliseconds() 913
- getMinutes() 914
- getMonth() 914
- getSeconds() 915
- getSelection() 696
- getTime() 686, 915
- getTimezoneOffset() 916
- getUTCDate() 916
- getUTCDay() 916
- getUTCFullYear() 917
- getUTCHours() 917
- getUTCMilliseconds() 917
- getUTCMinutes() 917
- getUTCMonth() 917
- getUTCSeconds() 917
- getYear() 917
- GIF 213
- GIF-Format 1111
  - Animation 1140
- Gitternetzlinien 243
- Gliederung 1148
- GMT 688, 909
- go() 892
- google 1156
- Google 1192
- gopher // 199
- Gopher 39, 199

grad 431  
 Grafik 850  
   als Verweis nutzen 227  
   ausrichten 219  
   ausrichten mit Tabellen 223  
   Breite 216  
   Höhe 216  
   Langer Beschreibungstext 218  
   Name zuweisen 218  
   Rahmen 217  
   Textfluss steuern 221  
   Verweis-sensitiv 228  
 Grafik einbinden 213  
 Grafik-Effekte  
   Dynamische Filter 1054  
 Grafikformate 1111  
 Graphics Interchange Format 1111  
 Graustufen 537, 1132  
 gray 103  
 green 103  
 Greenwich Mean Time 909  
 Greenwich-Zeit 688  
 greytex 429  
 groove 468  
 Größer-Als-Zeichen 101  
 groups 245  
 Guided tours 52  
 gutter 187  
  
**H**  
 h1-h6 729  
 HAM 54  
 handleEvent() 667, 697, 832, 841, 857, 873  
 hard 309  
 hasChildNodes() 789  
 hash 893  
 head 730  
 Header 93  
 HeaderName 1202  
 headers 268  
 hebrew 480  
 height 362, 501, 853, 858, 961  
 Helligkeit 1133  
 help 528  
 hidden 140, 325, 363, 468, 503, 509

hide 376  
 higher 523  
 highlight 429  
 highlighttext 429  
 Hintergrundbild 137, 251, 1119  
 Hintergrundbilder 471  
 Hintergrundfarbe 135, 251  
 Hintergrundfarben 471  
 Hintergrundmusik 140  
 Hintergrund-Sound 520  
 hiragana 480  
 hiragana-iroha 480  
 Histogramm 1134  
 history 891  
   back() 892  
   forward() 892  
   go() 892  
   length 892  
 Hit 1163  
 Holy Scrollers 53  
 home() 667  
 Homepage, Dateiname 112, 200  
 host 894  
 Host 30  
 Hosting-Provider 1150  
 hostname 894  
 Hostnummer 30  
 Hotkeys 211, 329  
 hr 730  
 href 231, 407, 895  
 hreflang 210  
 hspace 222, 260, 292, 354, 361, 363, 854, 859  
 htaccess 1195  
 html 731  
 HTML 42, 54, 73  
   Dokumenttyp-Angabe 95  
   Parser 93  
   Rellen zur Texteingabe 98  
   Syntax-Prüfung online 69  
 HTML 1.0 76  
 HTML 2.0 76  
 HTML 3.2 76  
 HTML 4.0 76  
 HTML 4.01 Transitional 96

HTML frameset 1273  
 HTML loose 1273  
 HTML prüfen, Online-Validator 115  
 HTML strict 1273  
 HTML transitional 1273  
 HTML-Datei  
     Grundgerüst 94  
     Titel 119  
 HTML-DTD 1273  
 HTML-Editoren 66  
 HTML-eigene Zeichen, Maskieren 101  
 HTML-Elemente 91  
 HTML-Elementobjekte 700  
 HTML-Parser 93, 389  
 HTML-Varianten 1273  
 HTML-Versionen 75  
 http// 109, 199  
 HTTP\_USER\_AGENT 1189  
 http-equiv 120  
 HTTP-Protokoll 42  
 https// 199  
 HTTP-Statuscodes 1173  
 Hurenkinder 516  
 HyperBase 54  
 Hyperlinkfarbe 682, 688  
     besuchte Seiten 690  
 Hyperlinks 75, 189  
 Hypertext 45, 113  
 Hypertext Abstract Machine 54  
 Hypertext Markup Language 42, 73  
 Hypertext Preprocessor 85  
 Hypertext Transfer Protocol 42  
 Hz 431

## I

i 732  
 IAB 35  
 IANA 35, 1183  
 ICab 1192  
 ICANN 34, 35  
 id 267, 378, 422, 801  
 IDREF 1226  
 IETF 35, 62  
 iframe 732  
 image 334, 1179

Image Map 228  
 image/gif 1111  
 image/jpeg 1112  
 image/png 1113  
 image/svg+xml 1115  
 image/x-wavelet 1114  
 images 850  
     border 852  
     complete 852  
     handleEvent() 857  
     height 853  
     hspace 854  
     length 854  
     lowsrc 854  
     name 855  
     src 855  
     vspace 856  
     width 857  
 img 219, 733  
 implied 1225  
 in 427  
 inactiveborder 429  
 inactivecaption 429  
 Includes 1200  
 IncludesNOEXEC 1200  
 index.htm 112, 200, 390  
 index.html 112, 200  
 Indexes 1200  
 Indexmetapher 1148  
 indexOf() 968  
 IndexOptions 1203  
 IndexOrderDefault 1203  
 infinite 140  
 infobackground 429  
 Informations- und  
     Kommunikationsdienstegesetz 1169  
 Informationsgemeinschaft zur Festlegung der  
     Verbreitung von Werbeträgern e.V. 1163  
 Informationsverteilung 113  
 infotext 429  
 inherit 376, 515  
 inline 507  
 Inline-Element 176  
 Inline-Elemente 167, 172  
 Inline-Layer 367

inline-table 508  
 Innenabstand 460  
 innerHeight 653  
 innerHTML 801  
 innerText 784, 802  
 innerWidth 653  
 input 735  
 ins 737  
 insertAdjacentHTML() 811  
 insertAdjacentText() 812  
 insertBefore() 790  
 insertData() 790  
 inset 468  
 inside 480  
 Interlaced 1111  
 Intermedia-Studie 59  
 International Network Information Center 35  
 Internet Architecture Board 35  
 Internet Assigned Numbers Authority 35  
 Internet Draft 62  
 Internet Engineering Task Force 35, 62  
 Internet Explorer, Identifikation 1190  
 Internet Protocol 30  
 Internet Society 35  
 Internet-Protokoll 109  
 Internet-Zugang 1150  
 InterNIC 35  
 Intervall  
     abbrechen 662  
     einrichten 676  
 IP 30  
 IP-Adresse 30  
     dynamische Zuteilung 30  
 IP-Adressen aussperren 1199  
 irc 200  
 IRC 39  
 isFinite() 980  
 isindex 737  
 ismap 232  
 isNaN() 981  
 ISO 3166-1 1183  
 ISO 639-1 1183  
 ISO 639-2 1183  
 ISO 8859-1 99  
 ISO/IEC 10646 1102

ISO-8859 1096  
 ISO-8859-1 1098  
 ISO-8859-10 1101  
 ISO-8859-2 1098  
 ISO-8859-3 1099  
 ISO-8859-4 1099  
 ISO-8859-5 1099  
 ISO-8859-6 1100  
 ISO-8859-7 1100  
 ISO-8859-8 1100  
 ISO-8859-9 1101  
 ISO-Norm 8879 74  
 isTextEdit 802  
 italic 435  
 italics() 969  
 IuKDG 1169  
 IVW 1163

## J

Java 86  
 java: 345  
 Java-Applet, Parameter 346  
 Java-Applets 345  
     einbinden nach HTML 3.2 357  
 javaEnabled() 942  
 JavaScript 81, 1002  
     Event-Handler 379  
     Funktion 926  
     Mathematik 930  
     Sprachversion festlegen 126  
 JDBC DataSource Applet 1051  
 join() 902  
 Joint Photographic Expert Group 1112  
 Joyce, Michael 48  
 JPEG 213  
 JPEG-Format 1112  
     progressive JPEG 1113  
 JScript 82, 86, 884, 1002  
 justify 145, 453

## K

katakana 480  
 katakana-iroha 480  
 Kaufmännisches Und 101  
 kbd 738

keyCode 886  
 keywords 121  
 kHz 431  
 Kindknoten 987  
 KK-Antrag 1165  
 Klasse-A-Netz 30  
 Klasse-B-Netz 31  
 Klasse-C-Netz 31  
 Kleiner-Als-Zeichen 101  
 Klickbutton 319  
 Knoten 987  
 Kohäsive Geschlossenheit 50  
 Kommentar 107  
 Konnektierung 1165  
 Kontrast 1133  
 Kopfzelle 235  
 Kuhlen, Rainer 47

## L

label 316, 738  
 Label 327  
 landscape 512  
 lang 125, 377, 378, 395, 803  
 language 383, 803, 941  
 Language 1049  
 large 436  
 larger 436  
 lastChild 780  
 lastIndexOf() 969  
 lastModified 685, 688, 693  
 Latin-1 99, 1098  
 Latin-2 1098  
 Latin-3 1099  
 Latin-4 1099  
 Latin-5 1101  
 Latin-6 1101  
 Lauftext 184  
 Lautstärke 517  
 Layer 365, 862, 1012  
   Anzeigebereich beschneiden 368  
   Hintergrundbild 372  
   Hintergrundfarbe 371  
   Inline-Layer 367  
   Schichtposition 373  
   Sichtbarkeit 375  
   verschachteln 370  
 layers 367, 862, 1012  
   above 863  
   background 864  
   below 865  
   bgColor 864  
   captureEvents() 873  
   clip 865  
   document 867  
   handleEvent() 873  
   left 868  
   length 868  
   load() 874  
   moveAbove() 874  
   moveBelow() 875  
   moveBy() 875  
   moveTo() 876  
   moveToAbsolute() 877  
   name 869  
   pageX 869  
   pageY 869  
   parentLayer 870  
   releaseEvents() 877  
   resizeBy() 877  
   resizeTo() 878  
   routeEvent() 879  
   siblingAbove 870  
   siblingBelow 871  
   src 871  
   top 872  
   visibility 872  
   zIndex 873  
 layerX 887  
 layerY 887  
 Layout 261  
 Leerzeichen 99  
   geschützt 149  
   Geschütztes Leerzeichen 99  
   mehrere Leerzeichen erzwingen 149  
 left 145, 222, 361, 367, 453, 504, 506, 515, 522,  
   868  
 left behind 522  
 left: 496, 498  
 leftmargin 138  
 left-side 522

- left-side behind 522
  - leftwards 522
  - legend 739
  - length 680, 803, 823, 824, 826, 829, 846, 854, 859, 868, 881, 892, 900, 945, 948, 964
  - Lesezeichen 52
  - letter-spacing: 441
  - level 523
  - lhs 243
  - li 740
  - lighter 438
  - lime 103
  - line-height: 449
  - line-through 442
  - Linie 178
  - link 136, 741
  - link() 970
  - linkColor 688
  - links 879
    - length 881
    - location 880
    - name 880
    - target^ 881
    - text 882
    - x 882
    - y 882
  - Links 189
  - Listen 150
    - Definitionsliste 153
    - formatieren mit CSS 157, 477
    - HTML-Eigenschaften 155
    - Menüs und Verzeichnisse 154
    - Nummeriert 152
  - list-item 507
  - list-style: 482
  - list-style-image: 481
  - list-style-position: 480
  - list-style-type: 478
  - LiveWire 825
  - LN10 931
  - LN2 931
  - load() 874
  - location 880, 893
    - hash 893
    - host 894
    - hostname 894
    - href 895
    - pathname 895
    - port 896
    - protocol 896
    - reload() 897
    - replace() 897
    - search 896
  - location.href 320
  - locationbar 654
  - log() 935
  - LOG10E 932
  - LOG2E 931
  - Logarithmus 931, 932
  - Log-Dateien 1159
  - Logische Beziehungen 132
  - Logische Textauszeichnung 167
  - longdesc 219, 281, 292
  - loop 140, 363
  - loose 1274
  - Lost in Hyperspace 51
  - lower 523
  - lower-alpha 478
  - lowercase 443
  - lower-greek 480
  - lower-latin 478
  - lower-roman 478
  - lowsrc 215, 854
  - ltr 170, 378, 504
  - LuraWave-Format 1114
  - Lycos 1192
  - Lynx 1192
- ## M
- Mailing-Listen 70
  - mailto
    - zusätzliche Angaben 202
    - Sonderzeichen 204
  - mailto: 202, 828
  - male 524
  - map 742
  - margin: 459, 512
  - margin-bottom 227
  - margin-bottom: 456
  - marginheight 139, 283

- margin-left: 457
- margin-right 227
- margin-right: 458
- margin-top: 455
- marginwidth 139, 283
- Markenrecht 1168
- marker 507
- marks: 513
- maroon 103
- Marquee 184
- match() 970
- Math 929
  - abs() 933
  - acos() 933
  - asin() 933
  - atan() 934
  - ceil() 934
  - cos() 934
  - E 930
  - exp() 935
  - floor() 935
  - LN10 931
  - LN2 931
  - log() 935
  - LOG10E 932
  - LOG2E 931
  - max() 936
  - min() 936
  - PI 932
  - pow() 937
  - random() 937
  - round() 937
  - sin() 938
  - sqrt() 938
  - SQRT1\_2 932
  - SQRT2 932
  - tan() 938
- mathline: 448
- MathML 80
- Mauszeiger 527
- max() 936
- MAX\_VALUE 950
- max-height: 502
- maxlength 302, 323
- max-width: 501
- mayscript 359
- MdStV 1170
- media 382, 409
- Mediendienstestaatsvertrag 1170
- Medientyp 1179
- medium 436, 465, 523
- Mehrspaltiger Text 186
- Mehrzeilige Eingabefelder 305
- Memex 54
- menu 429, 743
- menubar 655
- Menüleiste 655
- Menülisten 154
- menutext 429
- message 1179
- meta 744
- Meta-Angaben 1065
  - Automatische Weiterleitung 127
  - Autor 121
  - Datum 121
  - Dublin Core 122
  - Grundlagen 119
  - Originaladresse 126
  - PICS-Rating 128
  - Profil 129
  - Schema 129
  - Scriptsprache 126
  - Sprache bei Stichwörtern 125
  - Stichworte 121
  - Style Sheets 126
  - Zeichensatz festlegen 125
- method 299, 830
- method= 299
- Microsoft, Entwickler-Infos 64
- MID 140
- middle 221, 353, 361, 451
- Milnet 28
- Mime 37, 1178
- Mime-Typ 343
- Mime-Type 207, 388
- Mime-Typen 1178
- mimeType 943
  - description 944
  - enabledPlugin 944
  - length 945



- suffixes 945
  - type 946
- min() 936
- MIN\_VALUE 951
- min-width: 500
- mix 521
- mm 427
- model 1179
- modifiers 887
- monospace 434
- Mosaic 43, 1192
- move 528
- moveAbove() 874
- moveBelow() 875
- moveBy() 667, 875
- moveTo() 668, 876
- moveToAbsolute() 877
- movie 350
- Mozilla 1189
- ms 431
- MSHTML Data Source 1054
- MSIE 1191
- Multimedia 341
- Multimedia-Objekt 858
- multipart 1179
- multiple 312
- Multipurpose Internet Mail Extensions 1178
- MultiViews 1200

## N

- n, m-Relation 50
- name 196, 278, 288, 290, 301, 318, 319, 320, 322, 347, 352, 359, 367, 655, 822, 831, 838, 855, 859, 869, 880, 949
- named entities 1263
- Namensraum 391
- NaN 951
- narrower 438
- National Center for Supercomputing 84
- National Science Foundation 28
- navigator 939, 1189
  - appCodeName 939
  - appName 940
  - appVersion 940
  - cookieEnabled 940
  - javaEnabled() 942
  - language 941
  - platform 941
  - userAgent 942
- navy 103
- NCSA 84
- NCSA-Server 1195
- NEGATIVE\_INFINITY 952
- Negativ-Effekt 540
- Nelson, Ted 47, 56
- ne-resize 528
- Netscape 43
  - Entwickler-Infos 64
- Netscape-Farbpalette 1129
- Netzwerknummer 30
- news: 199
- Newsgroup 28, 35, 40
- Newsgroups 70, 199
- Newsreader 40
- nextSibling 781
- Nicht-proportionale Schrift 163
- node 775, 987
  - appendChild() 785
  - appendData() 786
  - attributes 777
  - childNodes 778
  - cloneNode() 786
  - data 779
  - deleteData() 787
  - firstChild 780
  - getAttribute() 788
  - getAttributeNode() 788
  - hasChildNodes() 789
  - insertBefore() 790
  - insertData() 790
  - lastChild 780
  - nextSibling 781
  - nodeName 782
  - nodeType 782
  - nodeValue 783
  - parentNode 784
  - previousSibling 784
  - removeAttribute() 791
  - removeAttributeNode() 792
  - removeChild() 792

- replaceChild() 793
- replaceData() 793
- setAttribute() 794
- setAttributeNode() 795
- nodeName 782
- nodeType 782
- nodeValue 692, 703, 704, 783
- noframes 744
- Noframes-Bereich 272
- nohref 231
- noindex 124
- none 245, 479, 504, 506, 513
- no-repeat 475
- noresize 284
- normal 435, 454, 518
- Northernlight 1192
- noscript 745
- Noscript-Bereich 384
- noshade 179
- nowrap 249, 454
- n-resize 528
- NSF 28
- Number 949
  - MAX\_VALUE 950
  - MIN\_VALUE 951
  - NaN 951
  - NEGATIVE\_INFINITY 952
  - POSITIVE\_INFINITY 952
  - to\_string() 954
  - toExponential() 953
  - toFixed() 953
  - toPrecision() 954
- Number() 983
- Nummerierte Liste 152
- Nummerierung, automatisch 493
- nw-resize 528
- O**
- object 359, 746
- Objekte 341
  - Rahmen 350
- oblique 435
- off 309
- Öffentliches Recht 1169
- offsetHeight 804
- offsetLeft 804
- offsetParent 805
- offsetTop 806
- offsetWidth 806
- offsetX 888
- offsetY 888
- ol 747
- olive 103
- once 489
- onclick 380
- onClick 320, 380
- ondblclick 380
- onkeydown 380
- onkeypress 380
- onKeyPress 736
- onkeyup 380
- Online-Dienst 1151
- onmousedown 380
- onmousemove 380
- onmouseout 380
- onmouseover 380
- onmouseup 380
- opacity 532
- open() 651, 669, 697
- opener 652
- Opera 1192
- optgroup 748
- option 749
- options 842
  - defaultSelected 845
  - length 846
  - selected 847
  - selectedIndex 848
  - text 849
  - value 849
- Organisation 113
- orphans: 515
- outerHeight 656
- outerHTML 806
- outerText 807
- outerWidth 656
- outset 468
- outside 480
- overflow: 499, 502
- overline 442

**P**

- p 750
- Packer 68
- padding: 462
- padding-bottom: 461
- padding-left: 462
- padding-right: 462
- padding-top: 461
- page-break-after: 515
- page-break-before: 514
- Page-Impressions 1163
- Pageviews 1163
- pagex 371
- pageX 869, 888
- pageXOffset 657
- pagey 371
- pageY 869, 888
- pageYOffset 657
- palette 363
- panose-1: 447
- param 751
- parentElement 805, 807
- parentLayer 870
- parentNode 784
- parentTextEdit 808
- parse() 918
- parseFloat() 981
- parseInt() 726, 982
- Parser 93
- Passermarken 513
- password 1053
- Passwort, Eingabefeld 304
- Passwortschutz 1195
- Patentrecht 1168
- pathname 895
- Pause 518, 519
- pause: 519
- pause-after: 519
- pause-before: 518
- pc 427
- Pellow, Nicola 43
- Perl 83
- Perl-Interpreter 83
- personalbar 658
- Persönlichkeitsrecht 1169
- Pfad 52
- PGP 37, 1197
- PHP 84, 112
- physical 309
- PI 932
- PICS 128
- PICS-Label 128
- pitch: 524
- pitch-range: 525
- pixelDepth 961
- Pixelgrafik 1127
- platform 941
- Platform for Internet Content Selection 128
- Platzierung 261
- play() 861
- play-during: 520
- Plugin 207, 342
- plugins 946
  - description 947
  - filename 948
  - length 948
  - name 949
- pluginspage 363
- pluginurl 362
- PNG 213
- PNG-Format 1113
- pointer 528
- polygon 230
- pop() 903
- port 896
- Portable Network Graphic 1113
- Portnummer 109
- portrait 512
- position: 495
- Positionierung von Elementen 494
- POSITIVE\_INFINITY 952
- post 299
- PostScript 1095
- pow() 937
- Präformatierter Text 163
  - CSS 165
- pre 454, 752
- Pretty Good Privacy 37
- previousSibling 784
- print() 671

Printmedien 511  
 Produkthaftungsrecht 1169  
 Produktname 1190  
 profile 129  
 Progressive JPEG 1113  
 Projektverwaltung 1143  
 prompt() 671, 699  
 Proposed Recommendation 61  
 protocol 896  
 Provider, Wechsel 1165  
 Proxy-Agent 1192  
 Pseudoformate 424, 489  
 pt 427  
 purple 103  
 push() 903  
 Push-Technologien 32  
 px 427

**Q**

q 752  
 Quadratwurzel 932, 938  
 Quelle angeben 160

**R**

rad 431  
 Radiant 933  
 radio 318  
 Radiobuttons 317  
 Rahmen 463  
 Rahmendicke 464  
 Rahmenfarbe 466  
 Rahmentyp 467  
 random() 726, 766, 937  
 RARE 29  
 Rastergrafik 1127  
 RDF 120  
 RDS 1050  
 ReadmeName 1202  
 readonly 310  
 Rechtliche Aspekte 1166  
 Recommendations 61  
 recordNumber 809  
 rect 230, 510  
 red 103

Referenzieren 108  
   Absolute Pfadangaben 110  
   Relative Pfadangaben 110  
   Übergeordnetes Verzeichnis 111  
 referrer 689, 970  
 refresh 127  
 RegExp 955  
   \$[1..9] 957  
   exec() 958  
   test() 958  
 Reguläre Ausdrücke 955  
 rel 208  
 rel= 132, 133  
 relative 496  
 Relative Pfadangaben 110  
 releaseEvents() 672, 698, 877  
 reload() 897  
 Remote Data Service 1050  
 removeAttribute() 791, 813, 820, 993  
 removeAttributeNode() 792  
 removeChild() 792  
 repeat 474, 521  
 repeat-x 474  
 repeat-y 475  
 replace() 897, 971  
 replaceChild() 793, 794  
 replaceData() 793  
 Request For Comment 62  
 require 1197  
 required 1225  
 Réseaux Associés pour la Recherche Européenne  
   29  
 Réseaux IP Européens 29  
 reset 333  
 reset() 832  
 resizeBy() 673, 877  
 resizeTo() 673, 878  
 Resource Description Framework 120  
 rev 208  
 revealTrans() 1060  
 reverse() 904  
 RFC 62  
 RFC822 205  
 rgb(...) 428

RGBA-Modell 1129  
 RGB-Wert 102  
 rhs 243  
 richness: 525  
 ridge 468  
 right 145, 222, 361, 453, 504, 506, 515, 522  
 right behind 522  
 right: 498  
 right-side 522  
 rightwards 522  
 RIPE 29  
 Robot 1192  
     Indexierung verhindern 124  
 robots 124  
 robots.txt 1193  
 Rollbalken 658  
 round() 937  
 routeEvent() 674, 698, 879  
 Routing 34  
 Routing-Tabellen 34  
 row 268  
 RowDelim 1049  
 rowgroup 269  
 rows 245, 273, 305  
 rowspan 256  
 rtl 170, 378, 504  
 Rückbeziehungen 134  
 rules 245  
 Rundung 937  
 run-in 507

## S

s 431, 753  
 samp 754  
 Sandbox 87  
 sans-serif 434  
 Sättigung 1133  
 Scalable Vector Graphics 1115  
 Scharfes S 100  
 Schatten 539  
 Schattenwurf 534  
 scheme 129  
 Schnittmarken 513  
 Schnotz-Studie 59  
 Schnupp, Peter 47  
 Schrift  
     dicktengleich 163  
     nicht-proportional 163  
 Schriftart 181  
     downloadbar 1105  
 Schriftartendatei 444  
 Schriften, Downloadbare Schriften 133  
 Schriftfarbe 135, 181  
 Schriftformatierung 433  
 Schriftgröße 181  
 Schusterjungen 515  
 scope 268  
 screen 382  
 Screen 959  
     availHeight 959  
     availWidth 960  
     colorDepth 960  
     height 961  
     pixelDepth 961  
     width 962  
 screenX 889  
 screenY 889  
 script 754  
 Script-Bereiche 382  
 scroll 475, 503  
 scrollamount 185  
 scrollbar 429  
 scrollbar: 528  
 scrollbar-3dlight-color 529  
 scrollbar-arrow-color 529  
 scrollbar-base-color 529  
 scrollbar-darkshadow-color 529  
 scrollbar-face-color 529  
 scrollbar-highlight-color 529  
 scrollbars 658  
 Scrollbars 281  
 scrollbar-shadow-color 529  
 scrollbar-track-color 529  
 scrollBy() 674  
 scrollDelay 185  
 scrolling 282  
 scrollIntoView() 813  
 scrollTo() 675  
 search 896  
 search() 972

- Secure Shell 37
- Segal, Ben 42
- Seitengröße 511
- Seitenlayout 261, 511
- Seitenleiste 658
- Seitenränder 138, 512
- Seitenumbruch 514, 515
- select 756
- select() 842
- selected 313, 847
- selectedIndex 848
- Selektor 413
- semi-condensed 438
- semi-expanded 439
- separate 469, 486
- se-resize 528
- serif 434
- Server Side Includes 112
- Server-Hosting 1153
- Server-Push 32
- Server-seitige verweis-sensitive Grafik 232
- Server-Software 31
- Sessions 1164
- setAttribute() 729, 794, 811, 814, 820
- setAttributeNode() 795
- setDate() 918
- setFullYear() 919
- setHours() 919
- setInterval() 676
- setMilliseconds() 920
- setMinutes() 920
- setMonth() 921
- setSeconds() 922
- setText() 768
- setTime() 686, 922
- setTimeout() 677, 680, 684, 802, 1087
- setUTCDate() 923
- setUTCDay() 923
- setUTCFullYear() 923
- setUTCHours() 923
- setUTCMilliseconds() 923
- setUTCMinutes() 924
- setUTCMonth() 924
- setUTCSeconds() 924
- setYear() 924
- SGML 54, 74, 387
- shape 230
- Sharks 53
- shift() 904
- shiftKey 885
- show 376, 488
- showUI 1052
- siblingAbove 870
- siblingBelow 871
- silver 103
- sin() 938, 995
- Sinus 938
- size 182, 302, 311, 323
- size: 511
- slice() 905, 973
- slow 523
- slower 523
- small 436, 757
- small() 973
- small-caps 436
- smaller 436
- SMIL 90, 1116
- soft 308
- solid 468
- Sondertasten 883
- Sonderzeichen 100
- Sort 1049
- sort() 906, 991
- Sound einbinden 140
- sourceIndex 809
- Spacer 187
- Spaltensatz 186
- span 237, 758
- speak: 518
- speak-header: 488
- speak-numeral: 526
- speak-punctuation: 526
- speech-rate: 523
- spell-out 518
- Spider 1192
- splice() 906
- split() 973
- Sprachausgabe 517
- Sprachenkürzel 1183
- Sprechertyp 523

- Sprechgeschwindigkeit 523
- Sprechhast 525
- Sprechhöhe 524
- Sprechweise 525
- SQL 1051
- sqlStatement 1052
- sqrt() 938
- SQRT1\_2 932
- SQRT2 932
- square 157, 479
- src 214, 278, 290, 334, 362, 369, 385, 855, 860, 871
  - url(Datei) 445
- src: 447
- s-resize 528
- SSH 37, 68, 1153
- Standard Generalized Markup Language 74, 387
- standby 356
- start 157
- startx 532
- starty 532
- static 496
- Statistik 1189
- Statistiken 1161
- status 660
- statusbar 659
- Statuszeile 659
- stemh: 447
- stemv: 447
- Stichworte 121
  - Sprache festlegen 125
- Stimmgewalt 525
- stop() 677, 861
- Strafrecht 1168
- stress: 525
- strict 1273
- Strict 96
- strike 759
- strike() 974
- string 962
  - anchor() 964
  - big() 965
  - blink() 965
  - bold() 965
  - charAt() 966
  - charCodeAt() 966
  - concat() 966
  - fixed() 967
  - fontcolor() 967
  - fontsize() 968
  - fromCharCode() 968
  - indexOf() 968
  - italics() 969
  - lastIndexOf() 969
  - length 964
  - link() 970
  - match() 970
  - replace() 971
  - search() 972
  - slice() 973
  - small() 973
  - split() 973
  - strike() 974
  - sub() 974
  - substr() 975
  - substring() 975
  - sup() 976
  - toLowerCase() 976
  - toUpperCase() 977
- String() 983
- Structured Generalized Markup Language 54
- style 378, 408, 426, 532, 760, 815
  - getAttribute() 819
  - removeAttribute() 820
  - setAttribute() 820
- style=1 532
- Stylesheets 77, 403
- sub 451, 761
- sub() 974
- Subdomain 1152
- subject 203
- submit 333
- submit() 832
- substr() 975
- substring() 975
- Subtyp 1179
- Suchfunktion 665
- Suchmaschine 1155, 1192
  - Indexierung verhindern 124
  - steuern mit robots.txt 1193

Stichwort angeben 121  
 suffixes 945  
 summary 266  
 sup 761  
 sup() 976  
 super 451  
 Support 69  
 SVG 80, 90  
 SVG-Format 1115  
 sw-resize 528  
 Symboleiste 660  
 Syntax prüfen, HTML-Validator 115

## T

### Tabelle

Ausrichtung 259  
 Außenrahmen 242  
 Breite angeben 245  
 definieren 233  
 dicktengleiche Schrift 163  
 Farbige Trennlinien 254  
 formatieren mit CSS 483  
 Fuß definieren 238  
 Gitternetzlinien steuern 243  
 Grafik ausrichten 223  
 Höhe angeben 245  
 Inhalt zusammenfassen 265  
 Kategorie definieren 269  
 Kopf definieren 238  
 Körper definieren 238  
 leere Zelle 235  
 Seitenlayout 261  
 Textfluss steuern 260  
 Überschrift 257  
 Unterschrift 257  
 Zellenabstand 240  
 Zellenbeschreibung 266  
 tabindex 211, 231, 328  
 table 508, 761  
 table-caption 508  
 table-column 508  
 table-column-group 508  
 table-footer-group 508  
 table-header-group 508  
 table-layout: 485

table-row 508  
 table-row-group 508  
 Tabular Data Control 1048  
 Tabulator 99, 210, 328  
 Tag, Grundlagen 91  
 tagName 808, 809  
 tan() 938  
 Tangens 938  
 target 194, 278, 288, 300, 831, 881  
 Tastaturkürzel 211, 329  
 Tastaturüberwachung 883  
 tbody 764  
 TCP 30  
 TCP/IP-Protokoll 28, 29  
 td 765  
 TDC 1048  
 TDDSG 298, 1170  
 TDG 1170  
 teal 103  
 Technical Reports 62  
 Teledienste-Datenschutzgesetz 298, 1170  
 Teledienstegegesetz 1170  
 telnet // 199  
 Telnet 37, 68, 199, 1153  
 test() 958  
 text 136, 528, 823, 849, 882, 1179  
 text/css 126, 382  
 text/html 388  
 text/javascript 126  
 text/xml 388  
 Textabsätze 144  
   ausrichten 145  
   formatieren mit CSS 146  
 text-align: 452  
 textarea 766  
 Textauszeichnung  
   logisch 167  
   physisch 167, 171  
 text-bottom 451  
 text-decoration 193  
 text-decoration: 442  
 Textfarbe 687  
 text-indent: 448  
 Textknoten 987  
 TextQualifier 1050



- Textrichtung ändern 170
  - text-shadow: 444
  - texttop 221, 353
  - text-top 451
  - Textumbruch 453
  - tfoot 768
  - th 769
  - The Internet Corporation for Assigned Names  
and Numbers 35
  - thead 770
  - thick 465
  - thin 465
  - Thread 41
  - threeddarkshadow 429
  - threedface 429
  - threedhighlight 429
  - threedlightshadow 429
  - threedshadow 429
  - Thumbnails 1123
  - Tim Berners-Lee 41
  - Timeout
    - abbrechen 663
    - einrichten 677
  - Titel 119, 689
  - title 231, 377, 378, 689, 771, 809
  - to\_string() 954
  - toExponential() 953
  - toFixed() 953
  - toGMTString() 686, 925
  - toLocaleString() 925
  - toLowerCase() 976
  - T-Online 1151
  - toolbar 660
  - top 197, 221, 353, 361, 367, 451, 872
  - top.close() 677
  - top: 497
  - Top-Level-Domain 32
  - topline: 448
  - topmargin 138
  - toPrecision() 954
  - toString() 982
  - toUpperCase() 977
  - transitional 1273
  - Transitional 97
  - Transmission Control Protocol 30
  - transparent 466
  - Transparente Farbe 1111
  - Transparenz 1130
  - Transparenz-Effekt 1130
  - Transparenzfarbe 534
  - Transparenzmaske 538
  - Trennlinie 178
  - true 908
  - TrueDoc 1106
  - TrueType 1095
  - tt 773
  - type 208, 301, 343, 382, 383, 838, 860, 890, 946
  - Typisierte Verweise 208
- U**
- u 773
  - Überblendeffekt 531
  - Überschriften 141
    - ausrichten 142
    - formatieren mit CSS 143
  - Uhrzeit 910
  - ul 774
  - ultra-condensed 439
  - ultra-expanded 439
  - Umkonnectierung 1165
  - Umlaute 100
  - undefined 661
  - underline 442
  - unescape() 984
  - Unicode 1095, 1102
  - unicode-range: 446
  - Uniform Resource Locator 109
  - Uniform Resource Name 109
  - units-per-em: 447
  - Universal Coordinated Time 909
  - Universal Resource Identifier 108
  - Universalattribut 93, 377
  - Universaleigenschaften 701
  - Unix-Dateinamen 111
  - unshift() 907
  - Upload 38, 322
  - upper-alpha 478
  - uppercase 443
  - upper-latin 479
  - upper-roman 478

Urheberrecht 108, 1168  
 URI 42, 108  
     zuletzt besuchte Seite 689  
 URL 109, 689  
 url([Datei]) 528  
 URN 109  
 UseHeader 1050  
 usemap 231, 345  
 Usenet 35, 40, 70, 199  
 user 1053  
 userAgent 839, 942  
 User-agent 1193  
 User-Agents 1189  
 userLanguage 941  
 UTC 169, 909  
 UTC() 925  
 UTC-Zeitangabe 121

## V

Validator 69, 115  
 valign 251  
 value 157, 304, 318, 319, 320, 322, 324, 325, 347,  
     359, 839, 849  
 value= 315  
 var 775  
 VBScript 86, 1002  
 Vektorgrafik 80  
 Verfasser 108  
 version 389  
 Version 1190  
 Versteckte Formularelemente 324  
 vertical-align 227  
 vertical-align: 450  
 Vertragsrecht 1168  
 Verweis  
     Grafik als V. 227  
     typisiert 208  
 Verweise 75, 189  
     Anker 196  
     auf beliebige Dateien 206  
     auf Frames verweisen 287  
     projekt-extern 198  
     projekt-intern 194  
     Tabulatorreihenfolge 210

    Tastaturkürzel 211  
     Zielfenster 193  
 Verweis-sensitive Grafik 228  
 Verwisch-Effekt 532  
 Verzeichnis, Referenzieren 111  
 Verzeichnislisten 154  
 Verzeichnismetapher 1148  
 Vester, Frederic 45  
 video 1179  
 virtual 309  
 Virtual Hosting 1152  
 visibility 376, 872, 1014  
 visibility: 509  
 visible 503, 509  
 Visits 1164  
 Visual Basic 1002  
 vlink 136  
 vlinkColor 690  
 voice-family: 523  
 void 243  
 volume: 517  
 vsides 243  
 vspace 222, 260, 292, 354, 361, 363, 856, 860

## W

W3C 35, 44, 60  
 W3-Konsortium 35, 44, 60  
 wais 200  
 wait 528  
 Wallpaper 137, 471, 1119  
 Wasserzeichen-Effekt 475  
 WAV 140  
 Wave() 1055  
 Wavelet-Kompression 1114  
 Web Embedding Fonts Tool 1107  
 web.de 1156  
 Web-Browser 66  
 Webseite, Titel 119  
 Webseite drucken 671  
 Web-Services 68  
 Webspace 1151  
 Web-Verzeichnis 1156  
 Web-Views 50  
 WEFT 1107

Weiterleitung 127  
 welcome.htm 112, 200  
 welcome.html 112, 200  
 Werbung 1155  
 Wettbewerbsrecht 1168  
 which 890  
 white 103  
 white-space: 453  
 White-Space-Zeichen 99  
 wider 438  
 widows: 516  
 width 237, 362, 367, 857, 860, 962  
 width: 499  
 window 429, 649  
     alert() 660  
     back() 661  
     blur() 661  
     captureEvents() 661  
     clearInterval() 662  
     clearTimeout() 663  
     close() 664  
     closed 652  
     confirm() 664  
     defaultStatus 652  
     disableExternalCapture() 665  
     enableExternalCapture() 665  
     find() 665  
     focus() 661, 666  
     forward() 666  
     handleEvent() 667  
     home() 667  
     innerHeight 653  
     innerWidth 653  
     locationbar 654  
     menubar 655  
     moveBy() 667  
     moveTo() 668  
     name 655  
     open() 669  
     outerHeight 656  
     outerWidth 656  
     pageXOffset 657  
     pageYOffset 657  
     personalbar 658  
     print() 671

    prompt() 671  
     releaseEvents() 672  
     resizeBy() 673  
     resizeTo() 673  
     routeEvent() 674  
     scrollbars 658  
     scrollBy() 674  
     scrollTo() 675  
     setInterval() 676  
     setTimeout() 677  
     status 660  
     statusbar 659  
     stop() 677  
     toolbar 660  
 windowframe 429  
 windowtext 429  
 WingDings 1095  
 winsock.dll 30, 1150  
 word-spacing: 440  
 Working Drafts 61  
 World Wide Web 41  
 wrap 308  
 w-resize 528  
 write() 698  
 writeln() 699  
 Wurzel 932  
 Wurzelement 94  
 WWW 41  
 WYSIWYG-Editor 66

## X

x 823, 882, 891  
 Xanadu 51, 56  
 x-fast 523  
 x-height: 447  
 XHTML 74, 387  
     Dateinamen 388  
     Dokumenttyp-Angaben 97  
     Groß-/Kleinschreibung 392  
     Leerraum in Wertzuweisungen 394  
     Mime-Type 388  
     Modularisierung 398  
     Namensraum 391  
     Script-Bereich 395  
     Style-Bereich 395

- Verschachtelungsregeln 396
- Verweise zu Ankern 394
- XHTML 1.1 398
  - Dokumenttyp-Angabe 98
- xhtml1-strict.dtd 390
- XHTML-Datei
  - Grundgerüst 94
- x-large 436
- xml, lang= 395
- XML 54, 74, 79, 387
- XML Data Source 1053
- XML-Deklaration 95, 389
- XML-Editor 67
- XML-Namensraum 95
- xmlns 391
- xmlns= 95
- XML-Parser 389
- XSL 80, 388, 404
- XSL Transformation 80
- x-slow 523
- XSLT 80
- x-small 436
- xx-large 436
- xx-small 436

**Y**

- y 824, 882, 891
- yahoo 1156
- yellow 103

**Z**

- Zähler 69
- ZapfDingbats 1095

- Zechensatz, Zeichenweise festlegen 100
- Zeichenketten 962
- Zeichenreferenz 1263
- Zeichensatz 99, 684, 687, 1093
  - Festlegen mit Meta-Angabe 125
  - Global festlegen 100
  - ISO 8859-1 99
  - Latin-1 99
- Zeilenhöhe 449
- Zeilenumbruch 99, 147
  - mehrzeilige Eingabefelder 307
  - verhindern 148
- Zeilenumbruch erlauben 149
- Zelle
  - Inhalt ausrichten 249
  - spaltenweise verbinden 254
  - zeilenweise verbinden 255
- Zellenabstand 240
- Zelleninnenabstand 240
- Zentrieren 175
- Zielfenster 193
- Zielfensterbasis 131
- Zielgruppe 1144
- z-index 375
- zIndex 873
- z-index: 506
- ZIP-Format 205
- Zitat 159
  - Quellenangabe 160
- Zitate, formatieren mit CSS 161
- Zivilrecht 1168
- Zugriffsstatistik 1160