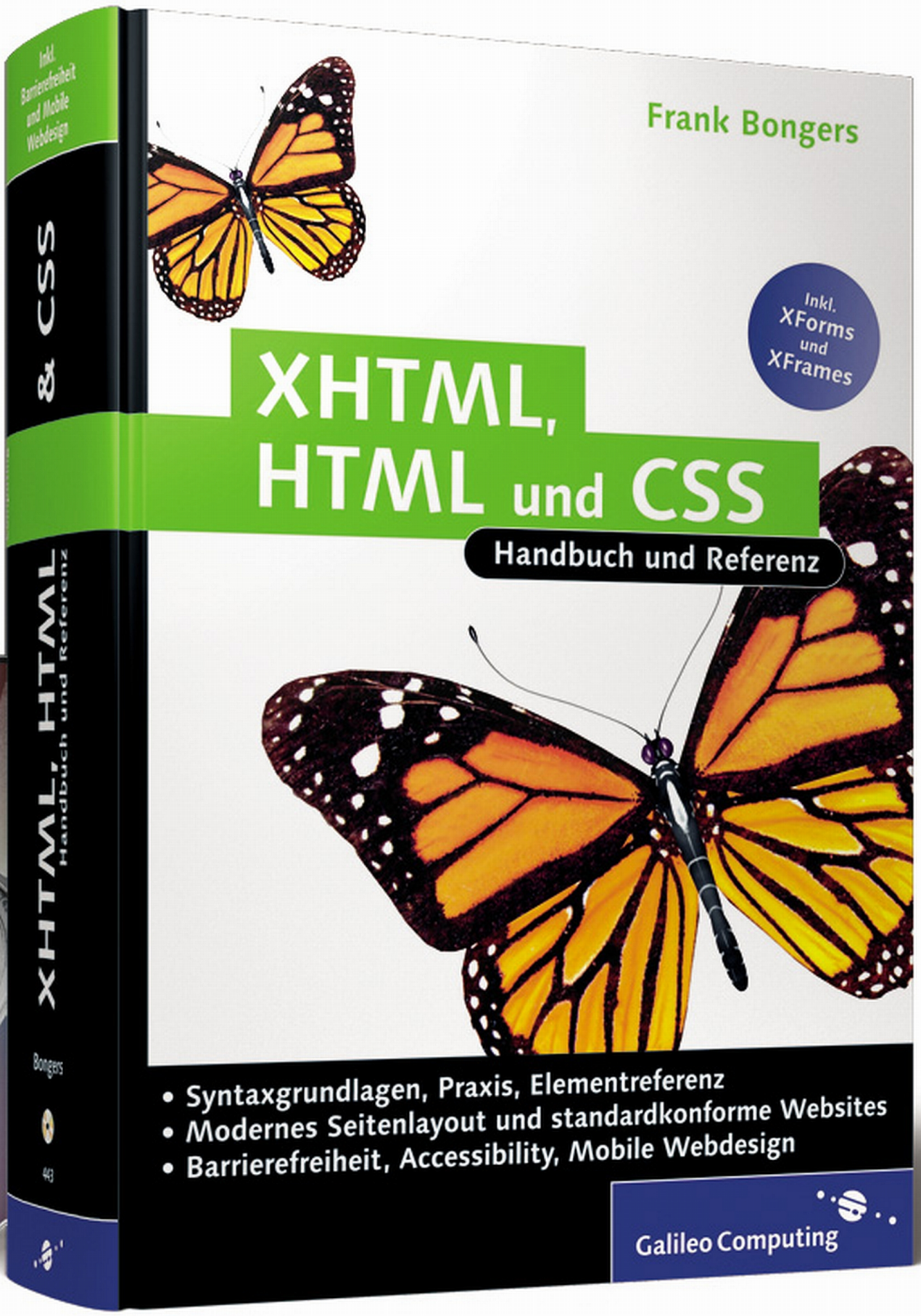
Inkl. Barrierefreiheit und Mobile Webdesign
& CSS
XHTML, HTML
Handbuch und Referenz
Bongers
443
Frank Bongers
Inkl. XForms und XFrames
XHTML, HTML und CSS
Handbuch und Referenz
• Syntaxgrundlagen, Praxis, Elementreferenz
• Modernes Seitenlayout und standardkonforme Websites
• Barrierefreiheit, Accessibility, Mobile Webdesign
Galileo Computing

Frank Bongers

XHTML, HTML und CSS

Handbuch und Referenz

Galileo Press

Liebe Leserin, lieber Leser,

jeder ernsthafte Webentwickler weiß um die Wichtigkeit solider HTML- und CSS-Kenntnisse. Auch der beste WYSIWIG-Editor (z. B. Dreamweaver oder GoLive) wird Ihnen die konsequente Arbeit am Quellcode nicht abnehmen können. Hier benötigen Sie ein Handbuch, das Sie bei der täglichen Arbeit unterstützt – beim Lernen, Arbeiten und Nachschlagen. Aus der Vielzahl an Büchern zum Thema sticht dieses Buch meiner Meinung nach deutlich heraus, denn es stellt Ihnen eine vollständige, umfassende und zugleich praxisnahe Beschreibung der aktuellen Möglichkeiten von XHTML, HTML und CSS mit all den zahlreichen Elementen und Attributen zur Verfügung.

Frank Bongers bietet Ihnen eine grundlegende Einführung in die (X)HTML- und CSS-Syntax und in die Gestaltung von Webseiten. Das Spektrum reicht dabei von der praktischen Umsetzung bis hin zum Webdesign für mobile Endgeräte oder Fragen zur barrierefreien Gestaltung von Webseiten. Den Hauptteil des Buches stellt der umfassende Referenzteil dar. Ideal für jeden Webdesigner und Entwickler, denn hier haben Sie erstmals eine vollständige Darstellung aller Elemente zur Verfügung. Als Lektor des Buches weiß ich, dass eine solche Leistung nicht mal so eben nebenbei zu erbringen ist. Umso mehr bewundere ich die Akribie und Detailgenauigkeit mit der Frank Bongers hier sein Wissen und seine Erfahrung ausbreitet. Ich bin sicher, dass dieses Handbuch schnell zu Ihrem unentbehrlichen täglichen Begleiter werden wird.

Dieses Buch wurde mit großer Sorgfalt lektoriert und produziert. Sollten Sie dennoch Fehler finden oder inhaltliche Anregungen haben, scheuen Sie sich nicht, mit uns Kontakt aufzunehmen. Ihre Fragen und Änderungsvorschläge sind uns jederzeit willkommen.

Ihr Stephan Mattescheck
Lektorat Galileo Computing

stephan.mattescheck@galileo-press.de
www.galileocomputing.de
Galileo Press · Rheinwerkallee 4 · 53227 Bonn

Auf einen Blick

1	(X)HTML & CSS – Einführung	21
2	(X)HTML – Die Syntaxgrundlagen	35
3	(X)HTML – die Dokumentstruktur	57
4	CSS Grundsyntax	85
5	Seitenlayout mit CSS	109
6	(X)HTML im Contentbereich	131
7	Navigationsgestaltung mit CSS	165
8	Code-Aktualisierung nach XHTML	195
9	Webdesign für mobile Clients	219
10	Barrierefreiheit und Accessibility	239
11	Werkzeuge für Webdesigner	271
12	HTML-Sprachfamilien	293
13	XForms und XFrames	359
14	(X)HTML-Elementreferenz	409
15	(X)HTML-Attributreferenz	915
16	Referenz HTML Entities	955
17	URL-Encoding und Language-Codes	971
18	Farben und Farbwerte	979
19	CSS-Referenz	991
A	W3C-Ressourcen	1137
B	Weitere Ressourcen	1165
C	Inhalt der Begleit-CD	1177

Der Name Galileo Press geht auf den italienischen Mathematiker und Philosophen Galileo Galilei (1564–1642) zurück. Er gilt als Gründungsfigur der neuzeitlichen Wissenschaft und wurde berühmt als Verfechter des modernen, heliozentrischen Weltbilds. Legendär ist sein Ausspruch *Eppur se muove* (Und sie bewegt sich doch). Das Emblem von Galileo Press ist der Jupiter, umkreist von den vier Galileischen Monden. Galilei entdeckte die nach ihm benannten Monde 1610.

Gerne stehen wir Ihnen mit Rat und Tat zur Seite:
stephan.matteschеck@galileo-press.de bei Fragen und Anmerkungen zum Inhalt des Buches
service@galileo-press.de für versandkostenfreie Bestellungen und Reklamationen
stefan.krumbiegel@galileo-press.de für Rezensions- und Schulungsexemplare

Lektorat Stephan Matteschеck
Fachgutachten Carsten Möhrke
Korrektorat Jürgen Dubau
Cover Barbara Thoben, Köln
Titelbild Fotolia
Typografie und Layout Vera Brauner
Herstellung Bernadette Blümel
Satz Typographie & Computer, Krefeld
Druck und Bindung Bercker Graphischer Betrieb, Kevelaer

Dieses Buch wurde gesetzt aus der Linotype Syntax Serif (9,25/13,25 pt) in FrameMaker.
Gedruckt wurde es auf fein holzhaltigem Naturpapier.

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

ISBN 978-3-89842-443-1

© Galileo Press, Bonn 2007
1. Auflage 2007

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion, der Vervielfältigung auf fotomechanischem oder anderen Wegen und der Speicherung in elektronischen Medien. Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Inhalt

Vorwort 17

1 (X)HTML & CSS – Einführung 21

1.1 Am Anfang war HTML 21
1.1.1 Eine sehr liberale Grammatik und ihre Vorteile 22
1.1.2 Verarbeitung und Ergänzung des Quellcodes durch den Parser 23
1.1.3 Die frühen Jahre von HTML 25
1.2 HTML 4.0 und CSS 26
1.2.1 HTML 4.0 und seine drei DTD-Varianten 26
1.2.2 Die Präsentationssprache CSS 27
1.3 Der Schritt von HTML zu XHTML 28
1.3.1 Ein einfaches XHTML-Dokument 29
1.4 Die Gegenwart von HTML und XHTML 30
1.4.1 Abwandlungen von HTML und XHTML 31
1.5 Der Schritt von CSS 1.0 zu CSS 2.1 31
1.6 Die Zukunft von HTML und XHTML 32
1.7 Die Zukunft von CSS 33
1.8 Zusammenfassung 33

2 (X)HTML – Die Syntaxgrundlagen 35

2.1 Grundsätzliches über SGML- und XML-Markupsprachen 35
2.1.1 Syntax von Start- und Endmarken 36
2.1.2 Elemente und ihre Inhaltsmodelle 36
2.1.3 Attribute – Metainformation über Elemente 38
2.1.4 Entitäten – allgemeine Ersetzungsvorschriften 39
2.1.5 Inhaltsmodelle von Elementen 41
2.2 HTML als SGML-Anwendung 43
2.2.1 Die Dokumenttyp-Definition 43
2.2.2 Elementvereinbarungen für HTML 43
2.2.3 Kommentare in HTML 46
2.2.4 Das Vokabular von HTML 47
2.3 XHTML als XML-Anwendung 48
2.3.1 Das Vokabular von XHTML 54
2.4 Zusammenfassung 55

3 (X)HTML – die Dokumentstruktur 57

3.1 Die Wahl der geeigneten Sprachversion 57
3.1.1 Möglichkeit 1: Korrekt geschriebenes HTML 57
3.1.2 Möglichkeit 2: Wohlgeformtes XHTML 58
3.1.3 Möglichkeit 3: Streng konformes XHTML 59
3.1.4 Ein paar Überlegungen zu XHTML 1.1 59
3.2 Grundaufbau von HTML- und XHTML-Dokumenten 60
3.3 Aufbau und Aufgaben des Dokumentkopfes 65
3.4 Aufbau und Aufgaben des Dokumentkörpers 69
3.5 Die Doctype-Deklaration und ihre Bedeutung 76
3.6 Beispiele für HTML-Grundstrukturen mit Doctype 79
3.6.1 HTML-Dokumenttypen 79
3.6.2 XHTML-Dokumenttypen 81
3.7 Zusammenfassung 83

4 CSS Grundsyntax 85

4.1 Die Grundsyntax von CSS 85
4.1.1 Verknüpfung von CSS-Anweisungen mit einem HTML-Dokument 85
4.1.2 Aufbau einer CSS-Anweisung 90
4.1.3 Die Deklaration einer CSS-Eigenschaft 91
4.2 Selektoren und Style-Einbindung 92
4.2.1 Der Selektor – Die Grundformen 92
4.2.2 Einbindung eines externen Stylesheets 98
4.2.3 Welcher Selektor – wann, wofür? 101
4.2.4 Kommentare in Stylesheets 102
4.2.5 Cascading, Addition und Konfliktregelung bei CSS-Anweisungen 103
4.2.6 Was bedeutet das »cascading« in Cascading Stylesheets? 103
4.2.7 CSS-Eigenschaften 104
4.3 Das CSS-Boxmodell 105
4.4 Zusammenfassung 107

5 Seitenlayout mit CSS 109

5.1 Grundüberlegungen zum Seitenaufbau 109
5.2 Umsetzung der Bereiche in Quellcode – Floatlayout 114

5.3 Quelltextreihenfolge und Accessibility 119
5.3.1 Die Wahl der Quelltextreihenfolge 119
5.3.2 Einbau der Sprunglinks 121
5.4 Umsetzung der Bereiche in Quellcode – Positionslayout 122
5.4.1 Schritt 1 – Festlegung der Containerbreiten 122
5.4.2 Schritt 2 – Platz für die Navigation schaffen 123
5.4.3 Schritt 3 – Positionierung des Navigationscontainers 124
5.4.4 Absolute und relative Positionierung 126
5.4.5 CSS-Eigenschaften für Positionierungszwecke 128
5.5 Zusammenfassung 128

6 (X)HTML im Contentbereich 131

6.1 Contentmarkup mit Textabsätzen und Überschriften 131
6.1.1 Mögliche globale Grundeinstellungen per CSS 131
6.1.2 Die Überschriften 134
6.1.3 Der Textabsatzcontainer 135
6.2 Inlineelemente zur Textauszeichnung 137
6.3 Der Hyperlinkcontainer, revisited 142
6.3.1 Welche Stylingmöglichkeiten bietet CSS für Hyperlinks? 145
6.3.2 Welche HTML-Attribute werden für Hyperlinks benötigt? 145
6.4 Eingebettete Medien 146
6.4.1 Welche Stylingmöglichkeiten bietet CSS für eingebettete Medien? 148
6.4.2 Welche HTML-Attribute werden für eingebettete Medien benötigt? 148
6.5 Listen 148
6.5.1 Welche Stylingmöglichkeiten bietet CSS für Listen? 151
6.5.2 Welche HTML-Attribute werden für Listen benötigt? 151
6.6 Tabellen 151
6.6.1 Welche Stylingmöglichkeiten bietet CSS für Tabellen? 155
6.6.2 Welche HTML-Attribute können für Tabellen verwendet werden? 155
6.7 Formulare 156
6.8 Zusammenfassung 163

7 Navigationsgestaltung mit CSS 165

7.1 Vorüberlegungen 165
7.1.1 Grundprinzipien der Navigationsgestaltung 166
7.1.2 Arten der Navigation 167
7.2 Navigationsmarkup 169
7.2.1 Horizontale Navigation mit Trennsymbolen 169
7.2.2 Vertikale Navigation mit Zeilenumbrüchen 170
7.2.3 Vertikale Navigation mit Textabsätzen 172
7.2.4 Vertikale Navigation mit Listen 173
7.3 Vertikale Navigation 174
7.4 Navigationspräsentation 176
7.4.1 Das Display-Property – Den Anker zur Fläche machen ... 176
7.4.2 CSS-Pseudoklassen – Mouse-Over-Effekte mit CSS 179
7.5 Vertikale verschachtelte Navigation 180
7.6 Horizontale Navigation 183
7.7 Karteireitermenü auf Listenbasis 187
7.8 Grafikgestützte Navigation auf Listenbasis 190
7.9 Zusammenfassung 192

8 Code-Aktualisierung nach XHTML 195

8.1 Codeaktualisierung mit HTML Tidy 195
8.1.1 HTML Tidy als Kommandozeilen-Tool 196
8.2 GUI-Versionen von HTML Tidy 206
8.2.1 Tidy UI für Windows 206
8.2.2 Tidy GUI für Windows 208
8.2.3 HTML Tidy in HTML-Kit 210
8.3 Codeaktualisierung von Hand 213
8.3.1 Praxisbeispiel eines 30 Minuten Redesigns 214
8.4 Zusammenfassung 218

9 Webdesign für mobile Clients 219

9.1 Welche Markup-Sprache für mobile Geräte? 219
9.2 Technische Randbedingungen mobiler Geräte 220
9.3 Displays von Mobilgeräten 221
9.4 Design für mobile Geräte 222
9.4.1 Schriftgröße 224
9.4.2 Quelltextaufbau 224
9.4.3 Farben 226
9.4.4 Navigation 227

9.5 CSS für mobile Geräte ... 228
9.5.1 Überlegungen zu einem Stylesheet für mobile Clients ... 229
9.6 Test von Mobil-Designs mit Opera ... 233
9.6.1 Test in Normalansicht ... 233
9.6.2 Test in Small-Screen Rendering-Ansicht ... 235
9.7 Test von Mobil-Designs mit Openwave Phone Simulator ... 236
9.8 Zusammenfassung ... 237

10 Barrierefreiheit und Accessibility ... 239

10.1 Barrierefreiheit – für wen? ... 239
10.1.1 Warum barrierefrei programmieren? ... 240
10.1.2 Technologien zur Überwindung von Barrieren ... 243
10.2 Das Einmaleins der Barrierearmut ... 244
10.2.1 Seitenaufbau und Layout ... 245
10.2.2 Verwendung von Schriften und Farben ... 247
10.2.3 Quelltext und Formatierung ... 249
10.2.4 Barrierefreie Tabellen ... 251
10.2.5 Links und Navigation ... 253
10.2.6 Barrierefreie Formulare ... 257
10.2.7 Bilder, Scripte und Multimedia ... 264
10.2.8 Weiterleitungen, Popups, dynamische Inhalte ... 267
10.2.9 Barrierefreie Inhalte ... 269
10.3 Zusammenfassung ... 269

11 Werkzeuge für Webdesigner ... 271

11.1 Editoren ... 271
11.1.1 Quelltexteditoren ... 271
11.1.2 WYSIWYG HTML-Editoren ... 274
11.2 Toolbars und weitere nützliche Werkzeuge ... 276
11.2.1 Developer Toolbars ... 276
11.2.2 Wave Toolbar ... 281
11.2.3 Accessibility-Tools ... 282
11.2.4 Bildschirmlineale ... 283
11.2.5 Colorpicker und Farbtools ... 284
11.3 Tools, Plug-ins und Erweiterungen ... 287
11.4 Textbrowser und Textbrowser-Simulationen ... 288

12 HTML-Sprachfamilien 293

12.1 Die Vergangenheit: HTML-Versionen 1.0 bis 3.2 293
12.1.1 HTML Level 0 293
12.1.2 HTML 1.0 295
12.1.3 HTML 2.0 295
12.1.4 HTML 3.2 297
12.2 Verpasste Chancen: HTML+ und HTML 3.0 298
12.2.1 HTML+ (HTMLPlus) 300
12.2.2 HTML 3.0 302
12.3 HTML 4.0 und HTML 4.01 303
12.3.1 HTML 4.01 strict 305
12.3.2 HTML 4.01 transitional 307
12.3.3 HTML 4.01 frameset 307
12.4 XHTML 1.0 und XHTML 1.1 308
12.4.1 XHTML 1.0 strict 309
12.4.2 XHTML 1.0 transitional 309
12.4.3 XHTML 1.0 frameset 310
12.4.4 XHTML 1.1 310
12.5 Bausteine: Die Modularisierung von XHTML 312
12.5.1 Der Common-Attributsatz 313
12.5.2 Core-Module 314
12.5.3 Das Strukturmodul 316
12.5.4 Das Textmodul 316
12.5.5 Das Hypertextmodul 318
12.5.6 Das Listenmodul 319
12.5.7 Das Präsentationsmodul (presentation module) 320
12.5.8 Das Formular-Basismodul (Basic Forms Module) 321
12.5.9 Tabellen-Basismodul (Basic Tables Module) 323
12.5.10 Formularmodul 325
12.5.11 Das Tabellenmodul 326
12.5.12 Das Image Modul 327
12.5.13 Das Applet Modul 328
12.5.14 Das Base Modul 328
12.5.15 Das Bi-Directional Text Modul 328
12.5.16 Das Client-Side Image Map Modul 329
12.5.17 Das Edit Modul 330
12.5.18 Das Frames Modul 330
12.5.19 Das iFrame Modul 331
12.5.20 Das Intrinsic Events Modul 331
12.5.21 Das Legacy Modul 332
12.5.22 Das Link Modul 335

12.5.23 Das Meta Information Modul 335
12.5.24 Das Name Identification Modul 336
12.5.25 Das Object Modul 336
12.5.26 Das Scripting Modul 336
12.5.27 Das Server-Side Image Map Modul 337
12.5.28 Das Style Attribut Modul 337
12.5.29 Das Style Sheet Modul 338
12.5.30 Das Target Modul 338
12.6 Die Hostlanguage: XHTML Basic 339
12.7 Die Zukunft: XHTML 2.0 340
12.7.1 Übersicht über die Elemente von XHTML 2.0 341
12.7.2 Modularisierung von XHTML 2.0 342
12.7.3 Allgemeiner Attributsatz der XHTML 2.0-Elemente 344
12.8 HTML ohne W3C: HTML 5.0 346
12.8.1 Die WHAT WG 346
12.8.2 »Web Applications« – auch »HTML 5.0« genannt 346
12.8.3 Geplante Erweiterungen für HTML 5.0 347
12.9 HTML für mobile Endgeräte 350
12.9.1 cHTML 350
12.9.2 iMode 1.0 352
12.9.3 iMode 2.0 352
12.9.4 iMode 3.0 353
12.9.5 iMode 4.0 354
12.9.6 Doti 1.0 354
12.9.7 HTML 4.0 mobile 355
12.10 XHTML für mobile Endgeräte 355
12.10.1 iMode XHTML 1.0 355
12.10.2 iMode XHTML 1.1 356
12.10.3 XHTML mobile profile 357

13 XForms und XFrames 359

13.1 XForms 359
13.1.1 Anwendungen und Implementierungen von XForms 360
13.1.2 Die Elemente von XForms 361
13.1.3 Der Namensraum von XForms 362
13.1.4 Die Module von XForms 362
13.1.5 Datentypen in XForms 368
13.1.6 XPath in XForms 368
13.1.7 XForms-Funktionen 369
13.1.8 Events in XForms 371
13.1.9 CSS-Selektoren für XForms 374

13.1.10 Aufbau eines XFormulars 375
13.1.11 Obligatorische und optionale Eingaben 378
13.1.12 Reglementierung der Eingabedaten durch XML-Schema 379
13.1.13 Versenden der Formulardaten – Der Submit-Button 384
13.1.14 Mehrfaches Submit 385
13.1.15 Mehrere Formulare 386
13.1.16 Beziehung zwischen Feldern und Dateninstanz 387
13.1.17 Beispiel 1: Ein- und Ausgabefelder in XFormularen 389
13.1.18 Beispiel 2: Eingabe eines Wertes innerhalb eines Wertebereichs 394
13.1.19 Beispiel 3: Eingabe eines Datums per XForm 396
13.2 XFrames 399
13.2.1 Die Elemente von XFrames 399
13.2.2 Beispiele für XFrames-Dokumente 404
13.2.3 Aufruf eines XFrames-Dokuments 406
13.3 Zusammenfassung 407

14 (X)HTML-Elementreferenz 409

14.1 a 411
14.2 abbr 430
14.3 acronym 433
14.4 address 437
14.5 applet 439
14.6 area 446
14.7 b 454
14.8 base 458
14.9 basefont 463
14.10 bdo 466
14.11 big 473
14.12 blockquote 476
14.13 body 481
14.14 br 493
14.15 button 498
14.16 caption 503
14.17 center 507
14.18 cite 510
14.19 code 512
14.20 col 515
14.21 colgroup 521
14.22 dd 527

14.23 del 530
14.24 dfn 533
14.25 dir 536
14.26 div 539
14.27 dl 549
14.28 dt 554
14.29 em 557
14.30 embed 560
14.31 fieldset 569
14.32 font 573
14.33 form 580
14.34 frame 592
14.35 frameset 604
14.36 h1 – h6 615
14.37 head 618
14.38 hr 621
14.39 html 626
14.40 i 633
14.41 iframe 635
14.42 img 643
14.43 input 655
14.44 ins 671
14.45 isindex 674
14.46 kbd 679
14.47 label 681
14.48 legend 685
14.49 li 688
14.50 link 693
14.51 listing 704
14.52 map 706
14.53 marquee 708
14.54 menu 715
14.55 meta 719
14.56 nextid 738
14.57 noframes 740
14.58 noscript 744
14.59 object 748
14.60 ol 759
14.61 optgroup 765
14.62 option 768
14.63 p 772

14.64 param 775
14.65 plaintext 779
14.66 pre 781
14.67 q 785
14.68 s 789
14.69 samp 791
14.70 script 794
14.71 select 803
14.72 small 809
14.73 span 812
14.74 strike 815
14.75 strong 817
14.76 style 819
14.77 sub 826
14.78 sup 828
14.79 table 830
14.80 tbody 845
14.81 td 851
14.82 textarea 863
14.83 tfoot 871
14.84 th 875
14.85 thead 887
14.86 title 892
14.87 tr 894
14.88 tt 900
14.89 u 903
14.90 ul 905
14.91 var 910
14.92 xmp 912

15 (X)HTML-Attributreferenz 915

15.1 Standardattribute in HTML und XHTML 915
15.2 Standard Eventhandler-Attribute 919
15.2.1 Eventhandler auf Applikations- und Interaktionsebene .. 920
15.2.2 Allgemein einsetzbare Eventhandler 921
15.3 Microsoft-Erweiterungen – Eventhandler 928
15.4 Microsoft-Erweiterungen – Standardattribute 947
15.4.1 Access-Attribute 947
15.4.2 Databinding-Attribute 949
15.4.3 Edit-Attribute 951

16 Referenz HTML Entities 955

16.1 HTML-Entities und Character-Entities 955

17 URL-Encoding und Language-Codes 971

17.1 URL-Encoding 971
17.2 Language Codes nach RFC 1766 974

18 Farben und Farbwerte 979

18.1 Farbnamen und Farbwerte 979

19 CSS-Referenz 991

19.1 Grundlagen 991
19.1.1 Formatierungsregeln 991
19.1.2 Position der Styledefinition 995
19.1.3 Präferenz und Cascading-Prinzip 999
19.1.4 Kommentare in Stylesheets 1002
19.2 CSS At-Rules 1003
19.3 CSS Selektoren CSS 1 und 2 1006
19.3.1 Attributabhängige Selektoren 1013
19.3.2 Pseudoelemente 1014
19.3.3 Pseudoklassen 1017
19.3.4 Pseudo-Kontextselektoren 1021
19.4 Werte und Maßeinheiten 1023
19.5 Properties CSS 1 und 2 1028
19.5.1 Textproperties 1028
19.5.2 Fontproperties 1038
19.5.3 Farben und Hintergründe 1044
19.5.4 Boxproperties 1053
19.5.5 Kontur-Properties 1067
19.5.6 Positioning-Properties 1072
19.5.7 Display-Property 1090
19.5.8 Listenstyle-Properties 1098
19.5.9 Tabellen-Properties 1103
19.5.10 Sonstige Properties 1111
19.5.11 Contentgenerierung 1114
19.5.12 Seitenformatierung (Druckausgabe) 1117

19.6 Browserspezifische Properties 1121
19.6.1 Scrollbar-Properties (Internet Explorer und Opera) 1121
19.7 Doctype-Switching und das CSS-Boxmodell 1127
19.7.1 Debugging von CSS 1132

Anhang 1135

A W3C-Ressourcen 1137
A.1 W3C-Spezifikationen 1137
A.1.1 W3C Ressourcen zu HTML 1139
A.1.2 W3C Ressourcen zu XHTML 1 1141
A.1.3 W3C Ressourcen zu XHTML 2 1145
A.1.4 W3C Ressourcen zu CSS 1 und 2 1146
A.1.5 W3C Ressourcen zu CSS 3 1147
A.1.6 Weitere CSS-Ressourcen 1155
A.1.7 W3C Ressourcen zu XML 1156
A.1.8 W3C-Ressourcen zu SVG 1158
A.1.9 W3C Ressourcen zu Accessibility 1160
A.1.10 Weitere W3C-Ressourcen 1161
B Weitere Ressourcen 1165
B.1 WHAT WG Initiative 1165
B.2 Mobiles Web 1166
B.3 Browser und Browserspezifikationen 1167
B.4 Webdeveloper-Tools 1169
B.4.1 Validatoren 1169
B.4.2 Toolbars 1170
B.4.3 Accessibility Tools 1171
B.5 Foren und Webzines 1172
B.6 Editoren 1173
B.6.1 Quelltexteditoren 1173
B.6.2 WYSISWYG-HTML-Editoren 1175
C Inhalt der Begleit-CD 1177
C.1 Quellcodebeispiele – Einführungskapitel 1177
C.2 Quellcodebeispiele – Referenzkapitel HTML 1177
C.3 Tools und Werkzeuge 1177
C.3.1 Hilfsprogramme 1177
C.3.2 Editoren 1177

Index 1179

Vorwort

Da es über mein Lieblingsthema HTML schon eine Vielzahl von Publikationen gab, sollte und musste dieses Buch »anders« sein als die bislang verfügbaren. Aus meiner langjährigen Erfahrung als Webdesigner und durch meine Unterrichtstätigkeit als Dozent war mir bewusst, dass es das komplette HTML-Buch bis heute nicht gibt.

Ein vollständiges Buch über HTML

Zusammen mit meinem Lektor kamen wir zu dem Schluss, dass es um eine wirkliche Vollständigkeit der Darstellung gehen sollte, die alle Aspekte von XHTML und HTML umfassen sollte. Auch CSS, ohne das im Bereich Webdesign heutzutage nichts mehr denkbar ist, sollte eingeschlossen werden. Hätte ich mir zu jenem Zeitpunkt eine Vorstellung von der mit diesen Anforderungen verbundenen Recherchearbeit gemacht, wäre dieses Buch vielleicht nicht in dieser Form geschrieben worden.

Ein praxisnahes Buch über HTML

Zahlreiche Entdeckungen und Wiederentdeckungen konnten wegen der (dann nun doch) beschränkten Seitenzahl nicht in dieses Buch gelangen – an dieser Stelle soll aber, zumindest ein einziges Mal, Netscapes geheimnisumwittertes Element `<hype>` erwähnt werden (es hat tatsächlich existiert). Natürlich war das Ziel dieses Buches nicht eine historische Abhandlung, sondern eine umfassende und praxisnahe Beschreibung der aktuellen Möglichkeiten, die XHTML und HTML in Form ihrer zahlreichen Elemente und Attribute bieten.

Aufgenommen in die ausführliche Referenz wurden HTML-Elemente nur dann, wenn einer der folgenden Punkte erfüllt war:

- Gültigkeit in einer der aktuellen Versionen von HTML oder XHTML
- Unterstützung in aktuellen Browsern
- Besondere Bedeutung in bestimmtem Praxiskontext

Aus diesem Grund werden neben den offiziell spezifizierten auch proprietäre Elemente wie `<embed>` oder `<marquee>` ausführlich beschrieben. Auch bei den Attributen wollte ich mich nicht auf die offiziellen beschränken. Gerade für den Internet Explorer existiert – Puristen mögen mir diese Aussage verzeihen – eine

Reihe von Erweiterungen, die höchst interessante Funktionalitäten bieten, über die es sich meiner Meinung nach lohnt, zumindest etwas zu erfahren.

Im Bereich CSS ist die Sachlage homogener – das Gewicht der Referenz im Rahmen dieses Buchs liegt auf CSS 2.1, da dies den Stand der aktuellen Browserunterstützung widerspiegelt. Eine Behandlung von CSS 3 ist zum gegenwärtigen Zeitpunkt noch müßig und unterblieb deshalb. Sie würde ohnehin ein eigenes Buch dieses Umfangs erfordern.

Ein einführendes Buch über HTML

Ohne eine anwendungsbezogene Verortung nützt eine Referenz jedoch wenig. Die ersten elf Kapitel beschäftigen sich daher mit praktischen Aspekten, die von einer grundlegenden Einführung in die HTML-Syntax und -Anwendung über Design mit CSS und XHTML bis hin zu Webdesign für Mobilgeräte reichen. Weitere Kapitel sind der Aktualisierung älterer Websites, der Barrierefreiheit sowie Tools und Werkzeugen gewidmet, die Ihnen als Webdesigner von Nutzen sein werden.

Ein Abriss über die Entwicklung der Sprachfamilien und Dialekte von HTML von den Anfängen bis hin zu XHTML 2.0 stellt den Auftakt zum umfassenden Referenzteil dar, der den Hauptteil des Buches bildet. In diesem Zusammenhang finden Sie auch eine Einführung in XForms und XFrames, die Ihnen einen Ausblick in die Zukunft der Formular- und Framesetgestaltung bietet. Auf die vollständigen Element- und Attributreferenzen folgen Abhandlungen, die HTML-Entitäten, URL-Encoding und Language Codes sowie Farben und Farbwerte betreffen. Den Abschluss des Buches bilden kommentierte Verzeichnisse der im Buch genannten Ressourcen und Spezifikationen.

Auf der Begleit-CD des Buches finden Sie neben den Quelltextbeispielen zu den Einführungskapiteln und der Referenz auch eine Reihe von nützlichen Tools und Editoren.

Die Icons in diesem Buch

Im Buch finden Sie viele Zusatzinformationen. Einige sind mit Icons gekennzeichnet, die Ihnen anzeigen, welcher Art die Information ist.

[!] *Achtung:* Hier werden Sie auf potenzielle Fehlerquellen aufmerksam gemacht.

[»] *Hinweis:* Diese Absätze enthalten hilfreiche Zusatzinformationen.

[zB] *Beispiel:* Hier finden Sie Beispiele, die Ihnen zeigen, wie Sie das dargestellte Wissen in der Praxis umsetzen können.

Danksagungen

Ein Buch mit einem Umfang wie diesem kann sich der Autor nicht allein auf die Fahne schreiben – ohne die aktive Unterstützung vieler anderer stünde er mit einem solchen Unterfangen auf verlorenem Posten. An dieser Stelle möchte ich allen danken, ohne die dieses Projekt nie zu Ende gebracht worden wäre. An erster Stelle meiner Frau Carina Prange, die mir mit viel Zuspruch über einen weit längeren Zeitraum als verabredet und absehbar den Rücken freihielt. An zweiter Stelle steht, wie immer, mein Lektor Stephan Matteschеck, der meinen wiederholten Terminüberschreitungen ein übermenschliches Maß an Geduld entgegensetzte und mit seinem kompetenten Rat dann doch alles zu einem guten Ende brachte. Ganz herzlich danken möchte ich auch meinem Fachgutachter Carsten Möhrke, der mich auf zahlreiche Unklarheiten in meinem Manuskript hinwies und so für deren Beseitigung sorgte und meinem Korrektor Jürgen Dubau, dessen wachsamen Auge kein falsch gesetztes Komma entging.

Last not least mein Dank an alle Leser dieses Vorworts, die bis hierher durchgehalten haben. Ich hoffe, dass dieses Buch, das Sie in den Händen halten, Ihnen nützlich sein wird.

Frank Bongers
Berlin

Wo kommt es eigentlich her, dieses HTML? Warum existiert inzwischen auch ein XHTML? Und was hat es mit CSS auf sich? Dieses Kapitel bietet Antworten in Form einer kurzen Einführung als Themenverortung – durchaus auch mit einem kleinen Blick hinter die technischen Kulissen.

1 (X)HTML & CSS – Einführung

In der heutigen Zeit, wo das Internet zur Selbstverständlichkeit geworden ist, kommt man kaum umhin, früher oder später mit HTML in Berührung zu geraten – immerhin handelt es sich hier um die Sprache, die jeder Webseite zugrunde liegt. Dabei ist diese Sprache, historisch betrachtet, gar nicht ausgesprochen alt: Erst etwa 1990 wurde sie vom Engländer Tim Berners-Lee am Schweizer CERN-Institut ins Leben gerufen. Sein Ziel bestand anfänglich nur darin, die Kommunikation unter den Mitarbeitern des Instituts über ein Rechnernetzwerk zu erleichtern. Ganz nebenbei erfand er dabei noch den Webserver und den Webbrowser, schrieb das Übertragungsprotokoll HTTP für Webseiten und prägte den Begriff »URI« (Uniform Resource Identifier).[1]

1.1 Am Anfang war HTML

HTML, die »Hypertext Markup Language«, wie Berners-Lee seine Schöpfung taufte, war auf **Einfachheit** getrimmt: Jeder in Frage kommende Client (sprich »Browser«) sollte in der Lage sein, ein derartiges Dokument darzustellen, ungeachtet der sonstigen Randbedingungen wie Betriebssystem oder Rechnerhardware. Die Art der Darstellung (»wie«) wurde der Anwendung überlassen, das Dokument sollte lediglich eine Informationsstruktur (»was«) beinhalten.

Informationsaustausch über Rechnernetzwerke war nicht eigentlich etwas Neues – immerhin gab es zu diesem Zeitpunkt seit längerem E-Mail und FTP –, doch wirklich neu war die »Killerapplikation« von HTML, nämlich die Möglichkeit, Verweise zwischen Dokumenten zu beschreiben. Dem Traum des globalen

1 Im Jahr 2004 wurde Berners-Lee für seine Verdienste von der britischen Königin Elisabeth II. zum Ritter geschlagen.

Hypertext-Netzwerkes[2] kam man schlagartig recht nahe. Obendrein waren diese Verweise höchst intuitiv zu bedienen, nämlich per Mausklick.

Unterscheidbare Bereiche des Dokuments werden mit **Marken**, so genannten »Tags« gekennzeichnet, wobei **Startmarken** (Starttags) und **Endmarken** (Endtags) für Bereiche gesetzt werden konnten. Die Marken der Bereiche (letztere werden auch als **Elemente** der Seite bezeichnet) werden mit Winkelklammern eingeleitet und beendet. Endmarken enthalten zusätzlich einen, dem dort wiederholten Markenbezeichner vorangestellten Slash. Das Prinzip war also so einfach wie einleuchtend:

`<Marke>` ... Inhalte des Elements ... `</Marke>`

Anmerkung: Der hier zugrunde liegende Begriff »Markup« kommt nicht ursprünglich aus dem Computerbereich, sondern diente als Bezeichnung für das Vorbereiten eines Manuskript für den Druck (»to mark something up«) – gemeint sind die handschriftlichen Randbemerkungen des Korrektors, die sich an den Setzer richteten und die gewünschte Umsetzung (wie »fett«, »kursiv«, »eingerückt«) der so markierten Bereiche der Vorlage verdeutlichen sollten.

1.1.1 Eine sehr liberale Grammatik und ihre Vorteile

Hierbei mussten nicht alle Dokumentbereiche explizit gekennzeichnet werden. Ein nach den sehr liberalen Regeln der Anfangszeit vollkommen gültiges HTML-Dokument sieht so aus (mehr ist in der Tat nicht erforderlich):

```
<title>Dies ist ein gueltiges HTML-Dokument</TITLE>
<P>Es enthaelt einen anklickbaren
<a HREF=anderes_dokument.html>Verweis</a>
auf ein anderes Dokument.<BR>
Eine neue Zeile am Ende.
```

Listing 1.1 01_liberal.html – Ein sehr liberal geschriebenes HTML-Dokument

Natürlich konnte man ein HTML-Dokument gleichen Inhaltes auch komplizierter schreiben – doch erforderlich war das nicht. Dass die Sprache dementsprechend einfach zu erlernen war und wegen der liberalen Handhabung auch unkorrekt formulierter Dokumente durch die Webbrowser eine **hohe Fehlertoleranz** bestand (und noch besteht!), mag über ihren Siegeszug entschieden haben.

Das »Feature«, das die Erlernbarkeit erleichterte, bestand darin, dass bestimmte Marken oder ganze Elemente gefahrlos weggelassen werden konnten. Ein ent-

2 Wie zuvor schon von Vordenkern wie Vannevar Bush, Ted Nelson oder Douglas Engelbart erdacht, propagiert und teilweise umgesetzt.

sprechend »intelligenter« Client sollte in der Lage sein, den Code dennoch zu interpretieren und fehlende Teile gegebenenfalls hinzuzufügen.

Ein solcher Client (der Webbrowser, der hier technisch gesprochen als »Parser« arbeitet) ergänzt das vorstehende HTML-Dokument durch weitere Bereiche, die nur »implizit« vorhanden sind. Erst nach dieser Vervollständigung macht er sich daran, das Dokument darzustellen (die jetzt in Folge angesprochenen Elemente werden in den nächsten Kapiteln ausführlich erläutert).

1.1.2 Verarbeitung und Ergänzung des Quellcodes durch den Parser

Unser in der Tat sehr rudimentär gestaltetes Ausgangsdokument wird von der Parserfunktion des Browsers um die fehlenden Bestandteile ergänzt. Um das gesamte Dokument wird hierfür als **Wurzelelement** ein Hüllencontainer gelegt (von »Container« spricht man, weil sich zwischen dessen Start- und Endmarke Inhalte befinden). Das Wurzelelement besitzt, wie die Sprache selbst, den Bezeichner »HTML«. Zusätzlich wird das Dokument in einen logischen **Dokumentkopf** und einen logischen **Dokumentrumpf** geteilt, wobei der `<TITLE>`-Container dem Dokumentkopf zugeschlagen wird. Der Kopfbereich wird in einen Container `<HEAD>` eingeschlossen, der Rumpf in einen Container `<BODY>`.

Das Ergebnis, das im Arbeitsspeicher des Browsers entsteht, sieht folgendermaßen aus (Ergänzungen fett hervorgehoben). Ohne hier technisch werden zu wollen – es handelt es sich um den vom Parser generierten »Zwischencode«:

```
<HTML>
  <HEAD>
     <TITLE>Dies ist ein gueltiges HTML-Dokument</TITLE>
  </HEAD>
  <BODY>
     <P>Es enthaelt einen anklickbaren
     <A HREF="anderes_dokument.html">Verweis</A>
     auf ein anderes Dokument.<BR>
     Eine neue Zeile am Ende.</P>
  </BODY>
</HTML>
```

Listing 1.2 Der durch den Parser ergänzte HTML-Quelltext von liberal.html

Dieser Zwischencode mitsamt den Ergänzungen wird an die folgende Formatierungsstufe weitergegeben und liegt der Darstellung des Dokuments im Browserfenster zugrunde (siehe Abbildung 1.1).

Hierbei werden die Marken selbst aus der Darstellung entfernt; der Browser verwendet sie lediglich als Hinweis, wie der Elementinhalt zu handhaben ist. So

wird der Text in <TITLE> in der Titelleiste des Browserfensters gezeigt, der in <A>-Tags eingeschlossene Bereich wird zum anklickbaren Hyperlink.

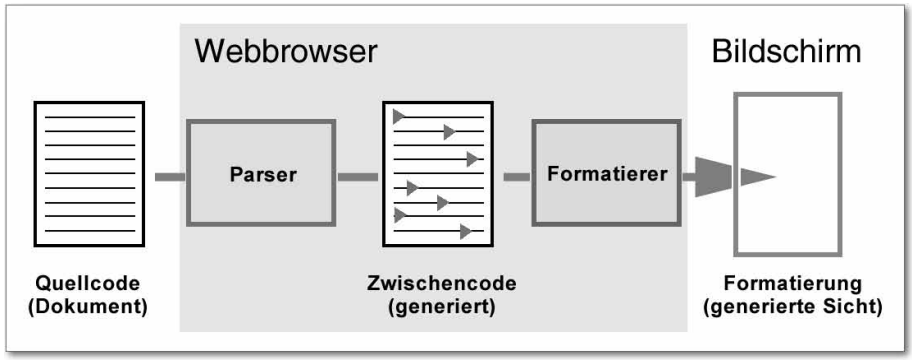

Abbildung 1.1 Der Webbrowser beinhaltet Parser und Formatierer.

Ein paar weitere Details werden im geparsten Quelltext sichtbar, nämlich die globale Änderung der Schreibweise der Elementnamen hin zu **Großbuchstaben**. Dies erlaubt dem Autor der HTML-Seite, die Bezeichner der Elemente im ursprünglichen Quelltext nach Belieben klein, groß oder gemischt zu schreiben.

Das Gleiche gilt für die Bezeichner so genannter »Attribute«, die Zusatzinformationen in Zusammenhang mit Elementen enthalten. Im obigen Quelltext ist dies der (dort klein geschriebene) Name des `href`-Attributs des Anker-Tags <A>: Auch dieser wird als `HREF` interpretiert. Beachten Sie, dass der Parser auch die **Anführungszeichen** um den Attributwert ergänzt, der den Dateinamen des Zieldokuments des Verweises enthält. Nicht verändert werden hingegen der Wert des Attributs und die Textinhalte der Elemente (hier der Dokumenttitel und der Textabsatz). Die **Einzelmarke**
 weist den Parser darauf hin, dass er an dieser Stelle in der Ausgabe einen Zeilenumbruch durchführen soll.

Die **Endmarke** </P> des Textabsatzes wird ergänzt und muss vom Autor nicht zwangsläufig gesetzt werden. Wollte er einen zweiten Textabsatz (englisch »paragraph«, daher der Elementname <P>) erzeugen, so müsste er allerdings für diesen zumindest eine Startmarke schreiben:

```
...
<p>Ein Textabsatz in HTML.
<p>Ein weiterer Textabsatz.
```

Dies wird vom Parser wiederum ergänzt zu

```
...
<P>Ein Textabsatz in HTML.</P>
<P>Ein weiterer Textabsatz.</P>
```

Die Option, Marken wegzulassen, stammt aus **SGML**[3], der Definitionssprache für Markupsprachen, von der Berners-Lee HTML ursprünglich abgeleitet hatte. Start- oder Endmarken von Bereichen (oder sogar beide), können hier als auslassbar (*omittable*) definiert werden. Diese Festlegung wird in einer Sprachgrammatik getroffen, welche die allgemeinen Regeln der Markupsprache enthält. Auch die Syntax dieser Sprachgrammatiken ist in SGML festgelegt.

Da es um die allgemeine Formulierung des Dokumenttyps »HTML« geht, spricht man von einer **Dokumenttyp-Definition** (kurz DTD, für *document type definition*). In dieser Dokumenttyp-Definition ist geregelt, welche Elemente existieren, welche Inhalte für sie zulässig sind und welche Attribute sie besitzen dürfen. Der Parser kennt diese DTD und verwendet ihre Regeln, um ein Dokument zu überprüfen und erforderliche Ergänzungen im Quelltext vornehmen.

1.1.3 Die frühen Jahre von HTML

Ganz zu Anfang jedoch gab es eine festgelegte Sprachgrammatik noch nicht – wie Berners-Lee erklärte, war HTML ursprünglich nur an SGML »angelehnt«, nicht aber von diesem korrekt abgeleitet. Dies geschah formal erst im Jahr 1994 anlässlich der Sprachversion **HTML 2.0**, der eine wirkliche Dokumenttyp-Definition zugrunde lag. Initiiert durch Berners-Lee und auf Grundlage der bisherigen Arbeitsgruppen, der Internet Enginering Task Force (IETF), wurde dann das **World Wide Web Commitee** (W3C) gegründet. Diese Institution sollte fortan HTML als Sprachstandard pflegen und weiterentwickeln.

Der durchschlagende Erfolg, der auf der Einfachheit von HTML beruhte, hatte in der Zwischenzeit zu einem Boom an Anwendungen und einer Explosion an Webseiten im sich entwickelnden Internet geführt. Von der bloßen Dokumentation von Information, möglichst ohne ihre Darstellung festzulegen (das war Berners-Lees Anfangsgedanke), verlagerte sich der Schwerpunkt in Richtung auf eine möglichst ansprechend präsentierte Optik der Webseiten.

Der Begriff »Webdesign« erschien auf der Bildfläche. HTML musste nun zunehmend Präsentationsaufgaben übernehmen, die das eigentliche Sprachkonzept zu verwässern drohten. Die Browserhersteller nutzten ihre Marktmacht, um die Webdesigner mit neuen HTML-Elementen zu »versorgen«, die reinen Präsentationszwecken dienten, und etablierten sie als Quasi-Standards, bis auch das W3C

3 Standard Generalized Markup Language, ISO-Standard 8879:1986; siehe http://www.iso.org.

sie zähneknirschend akzeptierte. Die Kompromissversion, auf die man sich Ende 1996 einigte, hieß **HTML 3.2**. Die Trennung von Struktur und Präsentation wurde ab diesem Zeitpunkt zunehmend verwässert und unterlaufen.[4]

Das »Feature« der Anfangstage, nämlich die sehr liberale Grammatik von HTML, entpuppte sich nun als Achillesferse. Zwar waren nicht alle Neuerungen jener Zeit per se »schlecht«, aber das Bedürfnis nach maschineller Verarbeitung von Informationen aus dem Internet stieg. Die Nutzergruppe erweiterte und verlagerte sich sozusagen – weg vom menschlichen Nutzer, der die Fähigkeit der Interpretation von Inhalten besitzt, hin zu auswertenden Computerprogrammen, die zur Verarbeitung eine genau geregelte Datenstruktur benötigen.

1.2 HTML 4.0 und CSS

Die Folgerung bestand darin, **Struktur** und **Präsentation** der Information wieder streng zu trennen. Die vom W3C propagierte Lösung bestand in zwei Sprachen, die jeweils ausschließlich einen der beiden Aspekte übernehmen sollten. Die eine davon war eine Neufassung von HTML, die andere nannte sich CSS.

1.2.1 HTML 4.0 und seine drei DTD-Varianten

Die Strukturaufgabe sollte ein »gereinigtes« HTML in einer neuen Version **HTML 4.0** übernehmen, das strikt auf Präsentationselemente verzichtete. Hierfür wurde als grammatische Grundlage eine radikal entmüllte, als **»strict«-DTD** bezeichnete Fassung erstellt. Diese Version entspricht der orthodoxen Auffassung des W3C von einer Markupsprache.

Ein vollständiger Bruch hätte jedoch die bis dahin erstellten Webseiten zur Ungültigkeit verdammt. Aus Gründen der Abwärtskompatibilität entschied man sich, für HTML 4.0 weiterhin eine **»transitional«-DTD** mit allen Präsentationselementen zur Verfügung zu stellen, sowie eine dritte Version, die **»frameset«-DTD**, mit der ein damals neues Feature, die HTML-Framesets, formuliert werden konnten. Die hier enthaltenen Elemente und Attribute, soweit sie nur Präsentationszwecken dienen, wurden jedoch als »deprecated« (missbilligt) gekennzeichnet und ihre zukünftige Entfernung vorbehalten.

Diese Spaltung gilt auch für das wenig später veröffentlichte Versionsupdate **HTML 4.01**, das die derzeit **aktuellste Sprachversion** von HTML darstellt.

4 Die sehr interessante Geschichte von HTML und der verschiedenen Versionen der Sprache ist in Kapitel 12, *Sprachfamilien von HTML*, genauer beschrieben.

Aufspaltung in drei Sprachversionen ab HTML 4.0

Seit 1997 zerfällt HTML in drei Sprachzweige:

- »transitional« – enthält Präsentationselemente
- »frameset« – enthält Präsentations- und Framesetelemente
- »strict« – die orthodoxe Variante ohne Präsentationselemente

1.2.2 Die Präsentationssprache CSS

Zur Formulierung von Darstellungsanweisungen wurde eine »Präsentationssprache« entworfen, die den Namen: **CSS** (Cascading Style Sheets) erhielt. Auch diese Idee war nicht neu; das W3C hatte sie bereits 1996 zu Zeiten von HTML 3.2 propagiert. Doch die Webdesigner wollten ungern von ihren mittlerweile gewohnten Präsentationselementen lassen, die Browserhersteller wiederum taten sich schwer mit der Unterstützung von CSS. Dennoch schien der Zeitpunkt für CSS nun günstig, da der Präsentationssprache jetzt in Form von HTML 4.0 *strict* eine reine Strukturspache gegenüber stand. Die Spezifikation **CSS 1.0** wurde daher 1999 in überarbeiteter Version erneut veröffentlicht.

Das Prinzip von CSS ist einfach: An die Bestandteile eines HTML-Dokuments, also die Elementcontainer und ihre Inhalte, werden **Darstellungsanweisungen** gebunden. Die Anweisungen betreffen Schriftart und -größe, aber auch Farbe, Hintergrund oder Abmessungen eines Elements. Die »Bindung« erfolgt über eine **Mustererkennung**, der im einfachsten Fall der Bezeichner eines Elements zugrunde liegt. Die folgende Anweisung gilt für `<p>`-Elemente:

```
p { font-family:arial; }
```

Die Anweisung besagt, dass der Inhalt aller `<p>`-Container des Dokuments mit dem Schrifttyp Arial darzustellen ist. Sie sehen hier bereits, dass die Syntax von CSS eine völlig andere ist als die von HTML. Dadurch, dass die CSS-Anweisungen im Idealfall gar nicht im HTML-Dokument zu stehen haben (ein Verweis genügt), stellt dies kein Problem dar.

Besteht diese Verbindung zwischen Element und Styleanweisung, so kann im HTML-Code auf Darstellungsanweisungen verzichtet werden:

```
<p>Dieser Textabsatz wird also mit Arial dargestellt.</p>
<p>Und dieser hier auch.</p>
```

Der Vorteil, auf diesem Wege Struktur (also HTML-Code) und Präsentation (also CSS-Anweisungen) vollkommen zu trennen, ist enorm. Bei der Verwendung von

HTML-Präsentationselementen dagegen müssen die Darstellungsanweisungen im HTML-Dokument selbst stehen, was etwa so aussehen würde:

```
<p><font face="arial">
Dieser Textabsatz wird mit Arial dargestellt.</font></p>
<p><font face="arial">Und dieser hier auch.</font></p>
```

Sie sehen, dass hier wesentlich **mehr Quelltext** im HTML erforderlich ist, als die CSS-Anweisung erfordert. Zudem muss die Präsentationsanweisung (diese wird mit dem `<font>`-Element vorgenommen) in jedem Absatz wiederholt werden. Es ist einleuchtend, dass Dokumente hierdurch erstens größer werden, zweitens schwerer zu warten sind (stellen Sie sich vor, Sie müssten die Schriftart beispielsweise überall auf Verdana ändern). Obendrein ist CSS mächtiger als die HTML-Präsentationselemente, was ein weiterer Grund ist, von ihrer Verwendung Abstand zu nehmen.

1.3 Der Schritt von HTML zu XHTML

Relativ früh kristallisierte sich heraus, dass HTML wegen seiner allzu liberalen Grammatik auf Dauer nicht tauglich bleiben würde. Um die Verarbeitung solcher Dokumente durch beliebige Anwendungen sicher zu gewährleisten, musste auf eine grundsätzlich **rigide Grammatik** aufgebaut werden. Da HTML bereits etabliert war, musste dies jedoch in einer Form erfolgen, die bestehende Browser verarbeiten können sollten – Abwärtskompatibilität war gefragt.

Die Lösung fand sich in Form der 1998 aus der Taufe gehobenen Markupsprache **XML**, die ähnlich wie SGML (von der sie im Übrigen ebenfalls abgeleitet ist) die Definition beliebiger weiterer Markupsprachen ermöglicht. In diesem Zusammenhang spricht man von »XML-Anwendungen«. Bereits 1999 erfolgte eine »Nachschöpfung« von HTML als XML-Anwendung: **XHTML 1.0**.

Aber Achtung: HTML und XHTML sind verschiedene Sprachen!

Da sich XML und SGML **unterscheiden**, gilt dies zwangsläufig auch für HTML und XHTML. Es handelt sich um einige Details, die jedoch nicht trivial sind. So bildet XHTML das Vokabular von HTML, das heißt dessen Elemente, deren Inhaltsmodelle und Attribute zwar weitestgehend nach. Doch bereits in der Grundsyntax von XML sind Unterschiede vorprogrammiert.

Ein wesentlicher Unterschied zwischen XHTML und HTML besteht darin, dass XML im Gegensatz zu SGML keine Verkürzungen zulässt, also das Weglassen von Marken. Ein XML-Dokument muss daher immer vollständig sein, weshalb eine Anwendung, die eine XML-Datei liest, wesentlich einfacher programmiert wer-

den kann. Auch ein XHTML-Dokument muss aus diesem Grund »komplett« sein – dieser Zustand wird auch als »wohlgeformt« bezeichnet.[5]

1.3.1 Ein einfaches XHTML-Dokument

Ein zum im Vorfeld vorgestellten »liberalen« HTML-Dokument gleichwertiges XHTML-Dokument ist wesentlich umfangreicher. Es besitzt, wie Sie sehen, auch eine Art »Vorspann«, der dazu dient, der verarbeitenden Anwendung mitzuteilen, welche Sprachversion (»Doctype«) genau hier vorliegt. Hier handelt es sich um die »transitional«-Version der XHTML-DTD.

Der Rest entspricht ansonsten dem Code, den der SGML-Parser im Zuge der durch ihn vorgenommenen Code-Ergänzung erzeugt hat. Für die Vollständigkeit des Quellcodes muss in diesem Falle der Autor selbst gesorgt haben – ein unvollständiges XHTML-Dokument gilt im Gegensatz zu einem (korrekt) verkürzten HTML-Dokument als fehlerhaft:

```
<!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>Dies ist ein gueltiges HTML-Dokument</title>
  </head>
  <body>
     <p>Es enthaelt einen anklickbaren
     <a href="anderes_dokument.html">Verweis</a>
     auf ein anderes Dokument.</p>
  </body>
</html>
```

Listing 1.3 Ein nicht gerade »liberales« XHTML-Dokument

Vielleicht fallen Ihnen noch weitere Einzelheiten auf – dass beispielsweise die Elementbezeichner hier konsequent klein geschrieben werden oder dass im Starttag `<html>` ein Attribut `xmlns` mit einer »merkwürdig anmutenden« Zeichenkette hinzugefügt wurde. Dies sind Eigenheiten von XHTML, auf die im folgenden Kapitel genauer eingegangen wird.

Ansonsten gilt:

- XHTML 1.0 behält alle Element- und Attributnamen aus HTML 4.01, allerdings in Kleinschreibung.

5 Verzeihen Sie mir an dieser Stelle die kleine Ungenauigkeit. Im nächsten Kapitel mehr dazu.

- XHTML 1.0 existiert wie HTML 4.01 ebenfalls in den Varianten »strict«, »transitional« und »frameset«.
- Weitere Unterschiede zwischen HTML und XHTML beruhen auf den Unterschieden zwischen den »Elternsprachen« SGML und XML.

1.4 Die Gegenwart von HTML und XHTML

Während HTML auf dem Versionsstand 4.01 seit 1999 quasi »eingefroren« weiterbesteht, blieb die Entwicklung von XHTML nicht vollständig stehen. In Form von **XHTML 1.1** wurde im Jahr 2001 die Aufteilung in drei Sprachvarianten ad acta gelegt – es existiert nur in einer Form, die den bisherigen *strict*-DTDs gleicht. Die Sprache wurde formal in so genannte »Module« aufgeteilt, die jeweils eine eng umrissene Funktionalität der Sprache umfassen und jeweils aus einer Untergruppe der Elemente besteht. Aus diesem Grund wird XHTML 1.1 auch als **modulares XHTML** bezeichnet.

Die Module dienen zur Ableitung anderer Markupsprachen für Spezialzwecke, die entweder auf HTML aufbauen oder Teile von HTML beinhalten sollen. Zum anderen wurde festgelegt, dass XHTML 1.1 **als XML zu behandeln** ist (dies hat Konsequenzen für die Handhabung der Dokumente durch den Server wie den Client). Das Gesamtvokabular wurde nur durch wenige Elemente (die »Ruby«-Elemente) erweitert, blieb aber ansonsten gegenüber XHTML 1.0 »strict« konstant. Auf die Frameset-Funktionalität wurde in XHTML 1.1 verzichtet.

Die neue Version ist jedoch nicht als Ablösung von XHTML 1.0 zu verstehen (oder hat sich zumindest nicht in diesem Sinne durchgesetzt), so dass im Augenblick folglich **sieben verschiedene offizielle Versionen von HTML und XHTML** parallel in Gebrauch sind (siehe Abbildung 1.2):

- drei Versionen von HTML 4.01 (»strict«, »transitional«, »frameset«)
- drei Versionen von XHTML 1.0 (»strict«, »transitional«, »frameset«)
- eine Version von XHTML 1.1 (entspräche einer »strict«-Version)

Nur die drei »strict«-Sprachversionen entsprechen der orthodoxen Interpretation von Markupsprachen. Die vier »transitional«- und »frameset«-Versionen von HTML und XHTML beinhalten hingegen Präsentationselemente und -attribute.

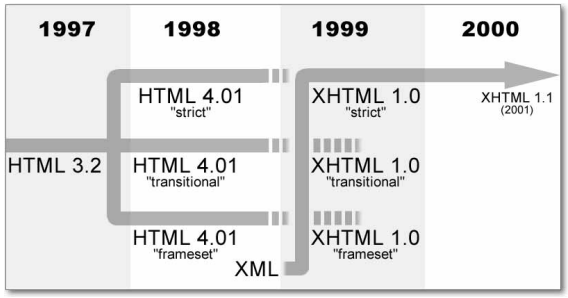

Abbildung 1.2 Die sieben aktuellen Varianten von HTML und XHTML

1.4.1 Abwandlungen von HTML und XHTML

Neben den »Großversionen« von XHTML existieren auch Abwandlungen (basierend größtenteils auf XHTML-Modulen), von denen die wichtigsten **XHTML Basic** (das »minimale« XHTML als so genannte »Host language«) und **XHTML mobile profile** sind. Letzteres ist eine vom W3C zwar abgesegnete, aber nicht initiierte Sprachversion für mobile Clients (Handys, PDAs). Vergleichbare, abgespeckte Versionen von HTML wie cHTML, HTML 4.0 mobile oder iMode HTML waren zum Teil proprietär und können inzwischen als veraltet gelten – doch deren Nachfolger iMode XHTML 1.0 und 1.1 allerdings nicht. Auf dem Mobilsektor zeichnet sich inzwischen jedoch eine Vereinheitlichung in Richtung auf XHTML mobile profile ab.

1.5 Der Schritt von CSS 1.0 zu CSS 2.1

Zu Beginn krankte **CSS 1.0** nicht an der Idee an sich, sondern an der mangelnden Unterstützung durch die Browserhersteller (es waren die Zeiten des »Browserkrieges«, wo man eher darauf achtete, sich voneinander abzusetzen, als allgemeinen Richtlinien zu folgen). Unverdrossen von den Aktualitäten hatte das W3C unterdessen die CSS-Spezifikation fortentwickelt und veröffentlichte 1998 die ambitioniertere und wesentlich umfangreichere Version **CSS 2.0**.

Eine Weile existierten zwei CSS-Versionen parallel, bis sich als unrealistisch herausstellte, von den Browserherstellern die Umsetzung von CSS 2.0 in angemessener Zeit zu erwarten. Man entschied sich stattdessen zurückzurudern – hier lässt sich eine Parallele zum geplanten HTML 3.0 und dem tatsächlich umgesetzten HTML 3.2 feststellen – und tilgte im Jahr 2002 aus CSS 2.0 einige der umstritteneren Bestandteile. Das Ergebnis wurde **CSS 2.1** getauft und stellt in der 2006 aktualisierten Fassung die derzeit aktuelle und auch weitestgehend unterstützte Version von CSS dar.

1.6 Die Zukunft von HTML und XHTML

Wie es genau mit HTML und XHTML weitergehen wird, scheint derzeit (Februar 2007) wieder einigermaßen offen. Sichtbar sind zwei konkurrierende Ansätze des W3C einerseits und eines Konsortiums namens **WHAT WG** andererseits, das sich aus Mitarbeitern von Browserherstellern und ehemaligen W3C-Mitgliedern zusammensetzt.

Die WHAT WG wirft dem W3C-Vorschlag für den HTML-Nachfolger »Weltfremdheit« vor und führt auch gute Gründe an. Dass hier Namen wie Håkon Wium Lie (Opera, CSS-Erfinder), Ian Hickson (Google) und Brendan Eich (Mozilla, JavaScript-Erfinder) vertreten sind, gibt dem Gegenvorschlag weiteres Gewicht. Unter der Bezeichnung **Web Application 1.0** propagiert die WHAT WG eine auf HTML aufbauende, abwärtskompatible Weiterentwicklung der Sprache. Diese soll parallel in einer HTML- und einer XHTML-Version existieren. Landläufig wird dieser Ansatz bereits als **HTML 5.0** bezeichnet.

Das **W3C** hat sich zu einer Neudefinition einer Markupsprache für das Internet entschlossen und diese als **XHTML 2.0** bezeichnet. Hierbei handelt es sich um eine nicht abwärtskompatible Neuschöpfung, die immerhin mit einigen Defiziten von HTML aufräumt und semantisch einen durchdachten Eindruck macht. Die Auslagerung der Frameset- und Formularfunktionalitäten in zwei externe Spezifikationen **XFrames** und **XForms** und das veränderte Vokabular von XHTML 2.0 machen jedoch eine völlig neue Browsergeneration erforderlich, die sich derzeit noch nicht abzeichnet. Obendrein gestaltet sich der Einigungsprozess schwierig – XHTML 2.0 existiert bereits in der achten Vorversion (Working Draft), ohne dass für die nahe Zukunft eine endgültige Spezifikation zu erwarten ist.

Zwischen beiden Fraktionen besteht ein Dialog, aus dem man eine Annäherung der Positionen ableiten kann. Inwiefern das W3C Teile des Vorschlags der WHAT WG für den HTML-Nachfolger übernehmen mag, ist spekulativ. Zumindest aber ist derweil der WHAT WG-Entwurf **Web Forms 2.0**, der in Konkurrenz zu **XForms** des W3C steht, bereits als W3C-Empfehlung aufgetaucht.

Weitere Informationen finden Sie hier:

- XHTML 2.0: **www.w3.org/TR/xhtml2**
- Web Application 1.0: **www.whatwg.org/specs/**
- Web Forms 2.0: **www.w3.org/TR/web-forms-2/**

1.7 Die Zukunft von CSS

Auch in Sachen CSS arbeitet das W3C seit 2001 an einer Weiterentwicklung, die unter der Bezeichnung CSS 3.0 firmiert. Analog zur Modularisierung von XHTML ist CSS 3.0 in verschiedene funktionale Module aufgespalten (es sind genauer gesagt 21 derartige Module), die jeweils von einer eigenen Arbeitsgruppe betreut werden.

Teile der geplanten Spezifikation sind in einigen Browsern vorwegnehmend bereits implementiert, von einer Fertigstellung als Standard ist CSS 3.0 indes noch weit entfernt. Dies mag an dem (wieder einmal) höchst ambitionierten Ansatz liegen, der einen enormen Umfang der Gesamtspezifikation zur Folge hat. Die Arbeit an den Modulen geht dabei unterschiedlich schnell voran – einige liegen seit mehreren Jahren brach, andere werden regelmäßig aktualisiert. Eine praktische Auseinandersetzung mit CSS 3.0 scheint zum gegenwärtigen Zeitpunkt noch nicht sinnvoll, die theoretisch sich abzeichnenden Möglichkeiten sind jedoch nichts weniger als faszinierend.

CSS aktuell: **www.w3.org/Style/CSS/current-work**

1.8 Zusammenfassung

Aus diesem Kapitel sollten Sie folgende Kernaussagen mitnehmen:

- Im Idealfall sollen Struktur und Darstellung von Information voneinander getrennt werden.
- Markupsprachen dienen der Kennzeichnung der Struktur von Dokumenten.
- Präsentationssprachen dienen der Beschreibung der Darstellung der Informationsstruktur.
- Dokumentteile werden durch Setzen von Marken (Tags) begrenzt, wobei zu Beginn eines Bereiches eine Startmarke und am Ende eine Endmarke gesetzt wird.
- Der Quellcode einer Markupsprache wird durch einen Parser verarbeitet, der das Ergebnis der Verarbeitung zur Darstellung weiterreicht.
- Für HTML-Dokumente geschehen Parsingvorgang und Darstellung innerhalb des Browsers.
- Ein SGML-konformer Parser ergänzt fehlende Marken. In SGML ist das Weglassen von Marken erlaubt, wo es die Grammatik des Dokumenttyps zulässt.

- HTML ist von SGML abgeleitet. In HTML dürfen bestimmte Marken daher weggelassen (verkürzt) werden. Weggelassene Marken werden durch den Browser vor der Darstellung ergänzt.
- Ein XML-konformer Parser ergänzt keine Marken, sondern meldet einen Fehler. Die Grundsyntax von XML erlaubt das Weglassen von Marken grundsätzlich nicht.
- XHTML ist von XML abgeleitet. In XHTML dürfen daher keine Marken weggelassen werden. Die daher obligatorische Vollständigkeit eines Dokuments erleichtert jedoch die Verarbeitung durch Software.
- HTML und XHTML dienen zur Formulierung der Struktur eines Dokuments. Beide Sprachen besitzen den gleichen Umfang an festgelegten Elementen und Attributen.
- CSS dient zur Formulierung von Darstellungsanweisungen für ein HTML-oder XHTML-Dokument. Die aktuelle Version CSS 2.1 wird weitestgehend unterstützt.
- HTML 4.0 und XHTML 1.0 existieren aus historischen Gründen in den drei Varianten »strict«, »transitional« und »frameset«. XHTML 1.1 existiert nur in einer einzigen Variante, die »strict« entspricht.
- Die als HTML 4.0 »strict« und XHTML 1.0 »strict« bezeichneten Varianten sowie XHTML 1.1 entsprechen der orthodoxen Vorstellung einer Markupsprache, die auf Präsentation verzichtet. Die anderen beiden Varianten enthalten Präsentationselemente, auf die besser verzichtet wird.

Wie geht es weiter?

Im folgenden Kapitel 2, *(X)HTML – die Syntaxgrundlagen,* wird näher auf die **Grundsyntax** von HTML und XHTML und die **Unterschiede** zwischen beiden Sprachen eingegangen. Sie erhalten auch einen (notwendigerweise knappen) Einblick in die Formulierung von SGML- und XML-konformen DTDs. Ebenfalls vertieft wird die **Syntax von CSS**, wobei auf die Formulierung der Darstellungsanweisungen und die Funktionsweise der Mustererkennung eingegangen wird, die eine Präsentationsanweisung mit einem Zielelement verbindet.

Thema dieses Kapitels sind die Syntaxgrundlagen von HTML und XHTML, ein Einblick in die Formulierung von DTDs sowie die Unterschiede zwischen beiden Sprachen.

2 (X)HTML – Die Syntaxgrundlagen

Im Volksmund werden HTML und XHTML oft als »Programmiersprachen« bezeichnet, was allerdings nicht zutrifft. Bei beiden handelt es sich vielmehr um Auszeichnungs- oder **Markupsprachen**. Während eine Programmiersprache einfach ausgedrückt Abläufe und Vorgänge beschreibt, dient eine Auszeichnungssprache dazu, die Struktur von in Textform vorliegender Information zu bezeichnen. Dieses Kapitel führt in die Syntaxgrundlagen erst von HTML, dann von XHTML ein und beschreibt die Unterschiede zwischen beiden Sprachen.

CSS ist im Gegensatz hierzu eine **Präsentationssprache**, welche die Darstellung von Inhalten beschreibt, indem sie Regeln hierfür aufstellt. Die Syntax dieser Regeln, ihre Verknüpfung mit den Elementen einer HTML-Seite und ihre Zusammenführung zu einem Stylesheet sind Thema des übernächsten Kapitels.

2.1 Grundsätzliches über SGML- und XML-Markupsprachen

Sowohl HTML als auch XML sind von SGML abgeleitet, XHTML wiederum von XML. Da die Sprachen demzufolge eng miteinander verwandt sind, gibt es **gemeinsame Grundlagen**. Diese sollen hier zunächst kurz skizziert werden, bevor auf die Eigenheiten von HTML und XHTML eingegangen wird.

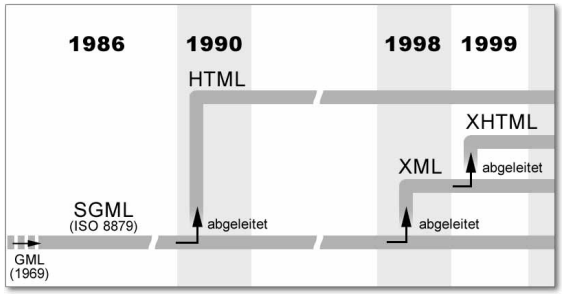

Abbildung 2.1 Abstammung von HTML und XML

2.1.1 Syntax von Start- und Endmarken

Vorhin ist bereits ein Stichwort gefallen – HTML wie auch XHTML sind **Textformate**. Es handelt sich also nicht um Daten in einem Binärformat, sondern um Daten in **ASCII-Code** oder einer vergleichbaren Codierung für Textformate. Dokumente in beiden Sprachen können also mit einem einfachen Texteditor erstellt werden.

Auch Markup-Befehle sind Text

Auch die **Auszeichnungsbefehle** der Markupsprache, die Marken oder »Tags«, wie sie gewöhnlich genannt werden, müssen als Text geschrieben werden. Damit sie vom eigentlichen Dokumentinhalt unterschieden werden können, müssen sie durch so genannte **Delimiter** begrenzt werden.

In SGML und demzufolge auch in HTML sowie in XML und XHTML dienen die **spitzen Klammern** < und > als Begrenzer eines Auszeichnungskommandos. Dazwischen steht der eigentliche **Name der Marke**, der »generic identifier«.

Die öffnende spitze Klammer zeigt als »open delimiter« den Beginn eines Auszeichnungskommandos an, die schließende spitze Klammer als »closing delimiter« (auch »tag close delimiter«) dessen Ende:

```
<name_der_marke>
```

Da mit den Auszeichnungen **Dokumentbereiche** begrenzt werden sollen, wird vereinbart, dass zum Anzeigen von Anfang und Ende eines Bereiches gleich bezeichnete Marken verwendet werden.

Um **Startmarke** und **Endmarke** zu unterscheiden, wird bei einer Endmarke vor den Bezeichner ein Slash gesetzt. Anders ausgedrückt besteht der »open delimiter« eines Endtags aus der Zeichenfolge »</«.

```
</name_der_marke>
```

2.1.2 Elemente und ihre Inhaltsmodelle

Den Bereich zwischen Start- und Endmarke bezeichnet man, *inklusive* der begrenzenden Marken, als **Element**. Zwischen den Marken können sich Inhalte befinden. Die folgende Abbildung 2.2 stellt abstrakt ein solches Element dar:

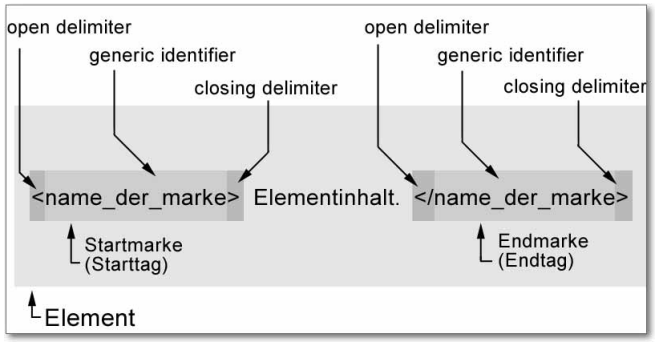

Abbildung 2.2 Aufbau eines Elements und seiner begrenzenden Marken

Der Inhalt eines Elements kann einfach aus Text bestehen, aber auch aus anderen Elementen oder einer Kombination aus Text und Elementen. Der für ein Element »erlaubte« Inhalt muss formal reglementiert werden. Die so festgelegte Form der Inhalte wird als **Inhaltsmodell** bezeichnet.

Verschachtelung von Elementen

Wenn ein Element ein anderes enthält, spricht man von einer **Verschachtelung** (Abbildung 2.3). Beachten Sie, dass das innere Element im äußeren Element *vollständig enthalten* ist:

```
<element_1>
  Dies sind Text und
     <element_2>
      ein weiteres Element
     </element_2>
  die als Inhalt eines Elements dienen.
</element_1>
```

Abbildung 2.3 Ein Element enthält Text und ein weiteres Element.

Eine **Teilüberlagerung** zweier Elemente würde zu einem nicht eindeutig zuzuordnenden Bereich führen und ist deshalb *verboten* (wie in Abbildung 2.4 der Text zwischen `<innen>` und `</aussen>`). Befindet sich die Startmarke eines Elements innerhalb eines anderen, so muss dies auch für die Endmarke gelten.

```
<aussen>
Elemente sind wie Schachteln.
Dinge sind entweder
      <innen> drin oder draußen.
       Jedenfalls solange, wie
                 </aussen>
      alles seine Richtigkeit hat!?
                          </innen>
```

Abbildung 2.4 Beispiel für unkorrekte Verschachtelung

Container

Im Vorfeld wurde es als Regel aufgestellt, dass ein Element über ein Start- und ein Endtag verfügt, zwischen dem sich Inhalte befinden. Ein Element, das diese Form besitzt und einen Dokumentbereich markiert, wird auch als **Container** bezeichnet. In der Tat handelt es sich bei den meisten Elementen in HTML und XHTML um Container.

Leere Elemente

Nicht alle Markup-Befehle jedoch müssen Dokumentbereiche kennzeichnen, sie können auch lediglich Anweisungen an den verarbeitenden Parser darstellen, an dieser Stelle des Verarbeitungsvorgangs etwas zu tun (beispielsweise in der Ausgabe einen Zeilenumbruch zu erzeugen).

Es genügt folglich, hier nur eine Startmarke zu setzen, da ja nichts begrenzt wird. Auch ein solches Markup ist vollständig, stellt also *per Definition* ebenfalls ein Element dar. Weil jedoch kein Inhalt vorhanden ist, spricht man von einem **leeren Element**.

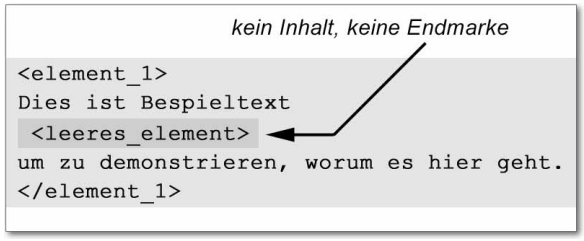

Abbildung 2.5 Ein leeres Element besitzt weder Inhalt noch eine Endmarke.

2.1.3 Attribute – Metainformation über Elemente

Soll über ein konkret im Dokument erscheinendes Element eine weitere Aussage in Form einer Metainformation gemacht werden, die also nicht Teil des Elementinhalts sein kann, so geschieht dies mit einem so genannten **Attribut**.

Abbildung 2.6 Ein Attribut trifft eine Aussage über »sein« Element.

Ein solches Attribut wird **im Starttag** notiert und gilt für den gesamten Elementbereich (Abbildung 2.6). Es besteht aus einem **Attributnamen** und einem **Attributwert**, der diesem mit Gleichheitszeichen zugewiesen wird. Damit der Parser den Beginn des Attributnamens erkennt, muss zwischen dem Elementnamen und dem Attributbezeichner mindestens ein **Leerzeichen** stehen.

In der Regel wird der Attributwert durch **Anführungszeichen** begrenzt, damit er prinzipiell auch Leerzeichen enthalten kann:

```
<tag beispiel="Ein Attributwert kann Leerzeichen enthalten">
```

2.1.4 Entitäten – allgemeine Ersetzungsvorschriften

Bevor auf die Inhaltsmodelle der Elemente eingegangen werden kann, muss zunächst ein weiteres Konzept umrissen werden, nämlich das der so genannten »Entitäten«. Dieses Konzept besitzt in HTML eine wesentlich breitere Bedeutung als in SGML, weshalb das Thema hier nur angerissen zu werden braucht.[1]

Entitäten als Maskierung von Sonderzeichen

In HTML werden Entitäten innerhalb des Textinhalts von Dokumenten eingesetzt. Bei HTML-Dokumenten handelt es sich, wie erwähnt, um ASCII-Textformate, denen daher von Haus aus nur eine begrenzte Zeichenmenge zur Verfügung steht. Werden **Sonderzeichen** wie Umlaute oder sprachspezifische Akzente benötigt, reicht der Umfang dieses Zeichensatzes nicht mehr aus.

Da erweiterte Zeichensätze das Risiko einer Fehlinterpretation in sich tragen, ist es wünschenswert, auch andere Zeichen durch ASCII-Codezeichen ausdrücken zu können. Dies geschieht mit Hilfe von **Ersetzungsvorschriften**, den Entitäten, die im Quelltext als »Platzhalter« für die gewünschten Zeichen dienen.

1 In SGML treten nicht nur Einzelzeichen, sondern ganze Textmodule als Entität auf. Auch in DTDs werden Entitäten eingesetzt, aber in einer leicht abgewandelten Form.

Eine **Entität** wird durch die Delimiter & und ; begrenzt, hat also die Form:

```
&name_der_entitaet;
```

Der Parser, der ein Dokument verarbeitet, ist in der Lage, eine Ersetzungsvorschrift durch das bezeichnete Zeichen zu ersetzen und in einer Form an die nachfolgende Anwendung weiterzugeben, die diese verarbeiten kann.

Steht beispielsweise die Entität `ä` als Ersatz für den Umlaut `ä` im Dokument, wird dies bei der Verarbeitung durch das Zeichen »ä« ersetzt. Dies ist in einer Form codiert, mit dem seine Verarbeitung sichergestellt ist. Das eigentliche Sonderzeichen »ä« braucht nicht im Quelltext zu stehen.

Entitäten als Maskierung von Steuerzeichen

Eine weitere Anwendung von Entitäten besteht in der **Maskierung** jener Zeichen, die den Charakter von **Steuerzeichen** besitzen. Hierzu zählen die spitzen Klammern, die als Delimiter für Marken dienen und das Et-Zeichen (Ampersand) selbst, das eine Entität anzeigt.

Müssen diese Zeichen an einer Position im Dokument erscheinen, an der sie als Steuerzeichen »missverstanden« werden können, so können sie durch Entitäten ersetzt werden. Der Parser reicht nach dem Ersetzungsvorgang die Zeichen »literal« an die Nachfolgeanwendung weiter, ohne dass sie als Steuerzeichen interpretiert werden.

SGML kennt »vordefinierte« Entitäten für Steuerzeichen

Weil diese Entitäten in jedem Fall zur Verfügung stehen müssen, sind sie in das Vokabular von SGML und XML fest eingebaut.

- `<` steht für die öffnende spitze Klammer < (»less than«)
- `>` steht für die schließende spitze Klammer > (»greater than«)
- `&` steht für das Et-Zeichen & (»ampersand sign«)
- `'` steht für das einfache Anführungszeichen ' (»apostrophe«)
- `"` steht für das doppelte Anführungszeichen " (»quotation mark«)

Mit Ausnahme von `'` gelten diese Entitäten auch in HTML.

Character-Referenzen – Entitäten mit Zeichencode

Stets möglich ist es, ein Zeichen durch einen Verweis auf seine Hexadezimal- oder Dezimalcodierung, den so genannten »character code« auszudrücken. Eine Entität, die sich dieser Methode bedient, nennt sich **Character-Referenz** (»character reference«). Anstelle eines Entity-Namens steht die Codezahl:

Das Copyrightzeichen © kann auf diese beiden Methoden umschrieben werden:

- `©` als dezimale Character-Referenz für ©
- `©` als hexadezimale Character-Referenz für ©

Beachten Sie im zweiten Fall das »x« als **Indikator für eine Hexadezimalzahl**. Vor der Codezahl steht in jedem Fall die **Raute #**. Ein zweites Beispiel demonstriert die Notwendigkeit einer Kennzeichnung:

- ` ` ist die dezimale Umschreibung eines Leerzeichens
- ` ` ist die hexadezimale Umschreibung eines Leerzeichens

Der Nachteil der Character-Referenzen gegenüber »normalen« Entitäten ist, dass man den Zeichencode des zu ersetzenden Zeichens kennen muss, statt auf einen »sprechenden« Entitäten-Namen zurückgreifen zu können.

2.1.5 Inhaltsmodelle von Elementen

Wie vorhin bereits erwähnt, kann ein Element Inhalte verschiedener Art besitzen. Welche dies im Einzelfall sind, wird in der DTD festgelegt. Man unterscheidet **vier Formen** von Inhalten.

Zwei davon bezeichnen reinen Textinhalt, wobei sich der Parser in Bezug auf im Text auftretende Entitäten unterschiedlich verhält:

- **CDATA-Textinhalt** (»character data«)
  Im Element darf **Text** erscheinen, aber keine anderen Elemente. Der Inhalt wird vom Parser *nicht* verarbeitet, kann also **keine Entitäten** beinhalten.
- **PCDATA-Textinhalt** (»parsed character data«)
  Im Element darf **Text** erscheinen, aber keine anderen Elemente. Der Inhalt wird vom Parser verarbeitet. **Entitäten werden** daher erkannt und **ersetzt**.

Am häufigsten anzutreffen sind »Elementinhalt« und »gemischter Inhalt«:

- **Elementinhalt**
  Es dürfen Elemente vorkommen, aber **nicht** unmittelbar **Text**.
- **Gemischter Inhalt**
  Es dürfen **beliebig** Text und weitere Elemente auftreten.

Beschreibung der Inhalte

Im Falle von gemischtem oder Elementinhalt werden die als Inhalt erlaubten Elemente und die Reihenfolge ihres Erscheinens genannt. Ansonsten wird zur Beschreibung ein **Schlüsselwort** verwendet.

Als Schlüsselworte gibt es in HTML nur zwei von Bedeutung:

- `CDATA` beschreibt den Inhalt des Elements als ungeparsten `CDATA`-Textinhalt (»character data«).
- `EMPTY` beschreibt das Element als »leer«. Jeglicher Inhalt ist verboten.

Soll der Elementinhalt aus geparstem Text bestehen, so wird dies folgendermaßen beschrieben (beachten Sie die runden Klammern und das Hash-Zeichen):

- `(#PCDATA)` beschreibt den Inhalt als geparsten PCDATA-Textinhalt (»parsed character data«)

Ansonsten werden **Elementnamen** genannt, die ebenfalls in **runde Klammern** gesetzt werden. Steht zwischen ihnen ein Komma, so bedeutet dies das Auftreten in der genannten Reihenfolge. Steht der senkrechte Trennstrich, so darf genau eines der Elemente ausgewählt werden:

- `(a,b,c)` bedeutet **die Folge** `<a><b><c>`
- `(a|b|c)` bedeutet **die Auswahl** `<a>` *oder* `<b>` *oder* `<c>`

Die Klammern dienen der Bildung einer **Inhaltsgruppe**. Ein optionaler Indikator hinter der Gruppe sagt aus, ob der durch sie beschriebene Inhalt wiederholbar ist, also mehrfach auftreten darf.

- `(Inhaltsmodell)`**?**
  Fragezeichen als Indikator: Der Inhalt ist optional, darf aber höchstens einmal auftreten (»kein oder einmal«).
- `(Inhaltsmodell)`
  Ohne Indikator: Der Inhalt muss erscheinen, darf aber nicht wiederholt werden (»genau einmal«).
- `(Inhaltsmodell)+`
  Pluszeichen als Indikator: Der Inhalt muss einmal erscheinen und darf dann beliebig oft wiederholt werden (»ein oder mehrmals«).
- `(Inhaltsmodell)*`
  Stern als Indikator: Der Inhalt ist optional, darf aber auch beliebig oft wiederholt werden (»kein oder mehrmals«).

[zB] *Beispiel:*

`(a,b)` erlaubt genau einmal die Folge `<a><b>`.

`(a,b)+` erlaubt `<a><b><a><b>` oder `<a><b><a><b><a><b>` etc.

`(a|b)` erlaubt `<a>` oder `<b>`.

`(a|b)*` erlaubt `<a><a><b>`, `<b><a><b>`, `<a><a><a><a>` oder »kein Inhalt«.

2.2 HTML als SGML-Anwendung

Eine komplette Einführung in die Schreibweise von DTDs nach SGML-Methode verbietet sich für dieses Buch (es ist auch so schon umfangreich genug). Hier sollen lediglich einige Grundlagen und ein paar Definitionen herausgegriffen werden, um exemplarisch das Prinzip der Deklarationen von HTML-Elementen zu erläutern. Im Anschluss wird ein Vergleich mit den Pendants in XHTML angestellt, die geringfügig anders deklariert werden. Mehr ist im Rahmen dieses Buches nicht erforderlich. Falls Sie allerdings Interesse an einer Vertiefung des Themas SGML haben, möchte ich auf entsprechende Fachliteratur verweisen.[2]

2.2.1 Die Dokumenttyp-Definition

Das Regelwerk wird insgesamt als **Dokumenttyp-Definition** (kurz **DTD**, für *document type definition*) bezeichnet. Alle nach einem Regelwerk produzierten Dokumente gelten (grammatisch gesehen) als vom gleichen Typ. Ein Parser, der ein solches Dokument erfolgreich verarbeiten soll, muss die DTD kennen.

Im Rahmen des **Regelwerks** für eine Markupsprache müssen folgende Einzelheiten festgelegt werden:

- Die **Namen der Elemente** und weitere Eigenschaften
- Das **Inhaltsmodell** jedes Elements
- Die erlaubten **Attribute** jedes Elements
- **Erlaubte Werte** der Attribute und weitere Eigenschaften
- Alle benötigten **Entitäten**

Um bei Bedarf die DTDs zu HTML lesen und verstehen zu können, hilft eine kurze Erläuterung ihrer Schlüsselbestandteile. Generell besteht eine DTD aus einer Folge von Definitionen, die als **Vereinbarungen** bezeichnet werden. Wie Vereinbarungen für Entitäten und Attribute getroffen werden, ist weniger wichtig – ein kurzer Blick soll jedoch auf die Vereinbarung für Elemente geworfen werden.

2.2.2 Elementvereinbarungen für HTML

An dieser Stelle ist es besser, konkrete Beispiele zu bringen, anstatt sich in abstrakten Darstellungen zu verlieren. In Abbildung 2.7 sehen Sie die für HTML gültige Vereinbarung des Wurzelelements `<HTML>`, das Sie aus dem vorherigen Kapitel bereits ansatzweise kennen.

2 Falls Sie dieses Thema vertiefen wollen, halte ich folgendes Buch für sehr geeignet (kurz und gut): Horst Szillat, *SGML – Eine praktische Einführung*; Thomson Publ. 1995.

Der **Elementname** wird am Anfang der Vereinbarung (gleich nachdem festgestellt wird, dass es sich um eine »Elementvereinbarung« handelt) bestimmt. In einer SGML-konformen Deklaration geschieht dies in Großbuchstaben. Da der Parser bei der Verarbeitung die Bezeichner gegebenenfalls entsprechend umschreibt, ist es im Dokument jedoch bedeutungslos, ob der Autor den Tagnamen als `<HTML>`, `<Html>` oder `<html>` schreibt (ich rate gleich an dieser Stelle aus verschiedenen Gründen zur Kleinschreibung).

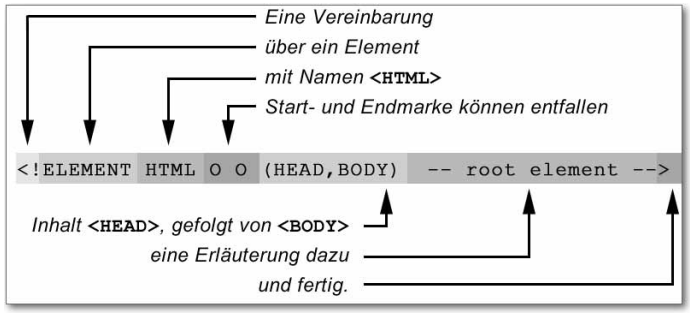

Abbildung 2.7 SGML-konforme Vereinbarung für das Element <HTML>

Für dieses Element darf der Autor beide Marken weglassen. Der Parser setzt sie in diesem Falle selbst ein. Als Inhaltsmodell enthält das Element `<HTML>` obligatorisch genau ein Element `<HEAD>`, gefolgt von genau einem Element `<BODY>`. Die Erläuterung (ein Kommentar) sagt aus, dass es sich um das Wurzelelement handelt. (Technisch hat ein solcher Kommentar keine weitere Auswirkung.)

Ebenfalls bekannt ist Ihnen bereits das Element `<TITLE>`, das den Dokumenttitel enthält. Dessen Vereinbarung sieht etwas anders aus.[3]

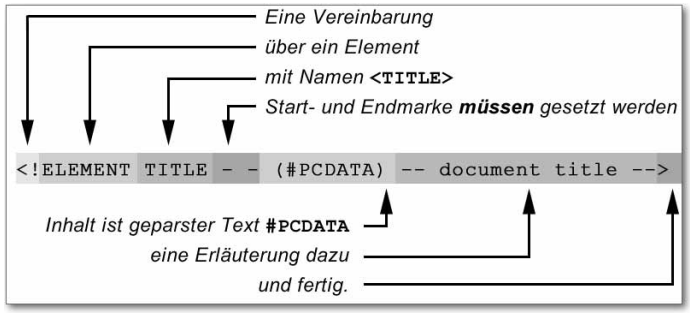

Abbildung 2.8 SGML-konforme Vereinbarung für das Element <TITLE>

3 Beide hier gezeigten Deklarationen sind aus didaktischen Gründen gegenüber den Originaldeklarationen leicht vereinfacht.

Für das <TITLE>-Element sind Start- und Endmarke Pflicht. Der Autor muss sie also stets einfügen. Der Parser könnte ihr Fehlen monieren. Im Dokumenttitel dürfen Entitäten erscheinen, die ersetzt werden. Der Elementinhalt ist deshalb als #PCDATA definiert.

Ein weiteres, bereits bekanntes Element ist der Textabsatzcontainer <P>. Sie haben gesehen, dass für diesen nur das Setzen der Startmarke erforderlich ist. Die Vereinbarung für dieses Element sehen Sie in Abbildung 2.9.

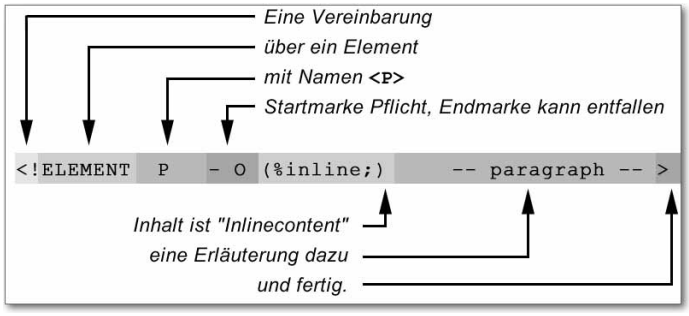

Abbildung 2.9 SGML-konforme Vereinbarung für das Element <P>

Bei dem Konstrukt %inline; welches das Inhaltsmodell beschreibt, handelt es sich um eine spezielle Form von Entität, die in DTD-Vereinbarungen verwendet wird, ein **Parameter-Entity**. An dieser Stelle muss nicht darauf eingegangen werden. Lesen Sie es einfach als Abkürzung für »Text und im Zeilenkontext einsetzbare Elemente«. Formal handelt es sich also um »gemischten Inhalt«.

Da es sich beim Element <P> um einen Container handelt, handelt es sich bei der Auslassbarkeit der Endmarke nicht um ein Muss – sie darf gesetzt werden. Lässt man sie weg, ergänzt der Parser sie an der Stelle im Quelltext, wo er den Textabsatz für beendet hält – beispielsweise, weil ein neuer Textabsatz beginnt. Dass dies die Möglichkeit einer Fehlinterpretation impliziert, dürfte auf der Hand liegen. Aus diesem Grund ist anzuraten, solche optionalen Marken stets physisch einzufügen.

Anders stellt sich die Lage bei leeren Elementen wie dem Zeilenumbruchbefehl <BR> dar. Die Vereinbarung ähnelt zwar oberflächlich jener für den Textabsatz <P>, jedoch hat die Auslassungsvereinbarung wegen des hier anderen Inhaltsmodells eine von diesem verschiedene Bedeutung.

Ist ein Element als leer beschrieben, so darf die Endmarke nicht nur entfallen, sie *muss* sogar entfallen. Leere Elemente sind in HTML weniger häufig. Die wichtigsten sind das eben angesprochene <BR>, außerdem existieren <HR> (die horizon-

tale Trennlinie), <INPUT> (erzeugt ein Eingabefeld eines Formulars) und <IMG> (fügt ein Bild ein). Weitere leere Elemente treten im Dokumentkopfbereich auf.

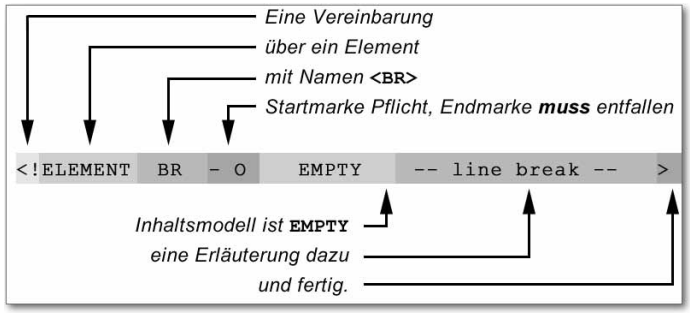

Abbildung 2.10 SGML-konforme Vereinbarung für das leere Element

Die Verkürzungsvereinbarung

Bei der SGML-konformen Vereinbarung eines Elements muss im Rahmen einer so genannten **Verkürzungsvereinbarung** festgelegt werden, ob kennzeichnende Marken entfallen können oder nicht. Muss eine Marke gesetzt werden, so wird dies mit einem Bindestrich »-« gekennzeichnet, darf sie entfallen, so steht ein großes »O« für »omittable« (auslassbar).

- O O bezeichnet die Auslassbarkeit sowohl der Start- als auch der Endmarke des vereinbarten Elements.
- - – bezeichnet Start- und Endmarke als vorgeschrieben. Keine der beiden Marken darf entfallen.
- - O bezeichnet die Startmarke als obligatorisch und die Endmarke als auslassbar (bei leeren Elementen *muss* Letztere entfallen).

  Anmerkung: Die Möglichkeit, die Startmarke als auslassbar und die Endmarke als obligatorisch zu definieren, wird in HTML nicht verwendet.

2.2.3 Kommentare in HTML

Den SGML-konformen Elementvereinbarungen dürfen, wie vorhin gezeigt, Erläuterungen beigefügt werden, die auch als »Kommentare« bezeichnet werden. Sie werden mit zwei unmittelbar aufeinanderfolgenden Bindestrichen eingeleitet und beendet.

```
<!ELEMENT hallo O O -- hallo gibt es gar nicht -->
```

Um nicht versehentlich das Ende des Kommentars zu verursachen, ist die Zeichenfolge `[Format Quelltext] -- [\Format Quelltext]` als Teil des Kommentartextes nicht gestattet:

```
<!ELEMENT huhu O O -- das hier -- klappt nicht -->
```

Dies hätte für HTML wenig praktischen Nutzen, würde SGML nicht auch so genannte »leere« Vereinbarungen kennen, in denen eigentlich gar nichts vereinbart wird. Diese haben die Form `<!>` und haben, für sich gesehen, wenig Sinn. Sie können aber ebenfalls Kommentare beinhalten:

```
<!-- diese leere Vereinbarung enthält einen Kommentar -->
```

Das Praktische ist, dass diese Art von Vereinbarungen auch im Inneren von SGML- und XML-Dokumenten eingesetzt werden darf, somit in HTML und XHTML ebenfalls verwendet werden kann. Daher bezeichnet man (eigentlich nicht korrekt) das gesamte Konstrukt inklusive der Vereinbarung als **HTML-Kommentar:**

```
<!-- dies ist in HTML ein Kommentar -->
```

Solche Kommentare dürfen an jeder beliebigen Stelle im Quelltext auftreten. Der Browser beachtet sie nicht, so dass sie im Browserfenster nicht zu sehen sind.

2.2.4 Das Vokabular von HTML

In HTML 4.01 sind insgesamt **91 Elemente** offiziell festgelegt. Dies ist der Umfang der »frameset«-Version, die die meisten Elemente enthält. Die »transitional«-DTD enthält zwei Elemente weniger, die »strict«-DTD, die keine Präsentationselemente enthält, ist auf 77 Elemente nochmals verringert.

In der folgenden Tabelle sind alle spezifizierten Elemente alphabetisch gelistet:

<A> <ABBR> <ACRONYM> <ADDRESS> <APPLET> <AREA>
<B> <BASE> <BASEFONT> <BDO> <BIG> <BLOCKQUOTE> <BODY>   <BUTTON>
<CAPTION> <CENTER> <CITE> <CODE> <COL> <COLGROUP>
<DD> <DEL> <DFN> <DIR> <DIV> <DL> <DT>
<EM> <FIELDSET> <FONT> <FORM> **<FRAME> <FRAMESET>**
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> <IFRAME> <IMG> <INPUT> <INS> <ISINDEX>
<KBD> <LABEL> <LEGEND> <LI> <LINK>
<MAP> <MENU> <META>

Tabelle 2.1 Elemente von HTML 4.01 frameset

<NOFRAMES> <NOSCRIPT>
<OBJECT> <OL> <OPTGROUP> <OPTION>
<P> <PARAM> <PRE> <Q>
<S> <SAMP> <SCRIPT> <SELECT> <SMALL> <SPAN> <STRIKE> <STRONG> <STYLE> <SUB> <SUP>
<TABLE> <TBODY> <TD> <TEXTAREA> <TFOOT> <TH> <THEAD> <TITLE> <TR> <TT>
<U> <UL> <VAR>

Tabelle 2.1 Elemente von HTML 4.01 frameset (Forts.)

Falls Sie als HTML-Neuling an dieser Stelle angekommen sind, sollten Sie nicht über diese scheinbare Vielzahl erschrecken. In der Praxis kommt man mit einem Großteil dieser Elemente eher selten in Kontakt, einige wenige dagegen stellen den Hauptanteil der verwendeten Tags.

Als Nächstes soll der Blick von HTML nach **XHTML** gehen. Beide Sprachen dienen der gleichen Aufgabe, nämlich dem Erstellen von Dokumenten, die durch Webbrowser dargestellt werden können. Ihre verschiedenartige Herkunft sorgt jedoch bei aller gewollten Ähnlichkeit auch für Unterschiede. Diese sollen jetzt kurz erläutert werden.

2.3 XHTML als XML-Anwendung

Bereits die Überschrift sagt es aus – XHTML ist, da von XML abgeleitet, eine so genannte **XML-Anwendung**. Bei HTML handelt es sich hingegen um eine **SGML-Anwendung**. Das Grundprinzip der Marken, Elemente, Attribute und Entitäten ist jedoch dasselbe, und auch die Element- und Attributnamen sind in beiden Sprachen im Grunde gleich. Das ist Absicht, da ein Browser möglichst problemlos beide Arten von Dokumenten verarbeiten können soll.

Auf Anhieb lässt sich beispielsweise *nicht* mit Sicherheit unterscheiden, ob das folgende Codefragment HTML oder XHTML ist (man kann jedoch annehmen, dass es sich eher um XHTML handeln wird):

```
<p>Das ist ein Textabsatz. Er wird in
HTML und XHTML gleich geschrieben.</p>
```

Unterschied 1 – Keine Elementverkürzung

Um einigen der Unterschiede auf die Spur zu kommen, genügt es, sich die Elementdeklarationen von XHTML anzusehen. Es handelt sich hier um **XML-kon-**

forme Vereinbarungen. Ihre Syntax ist (auch das ist gewollt), wie Abbildung 2.11 zeigt, gegenüber den SGML-konformen Vereinbarungen stark vereinfacht.

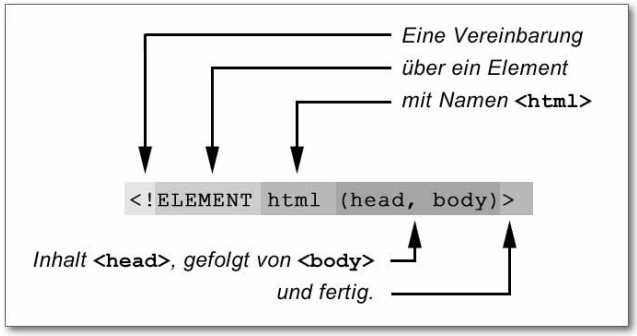

Abbildung 2.11 XML-konforme Vereinbarung für das Wurzelelement <html>

Erläuternde Kommentare in den Deklarationen entfallen in XML. Wenn es welche gibt, dann müssen sie außerhalb der Vereinbarungen stehen. Des Weiteren fällt beim Vergleich mit der SGML-konformen Vereinbarung auf, dass es **keine Verkürzungsvereinbarung** gibt.

Dies ist in einer XML-konformen Vereinbarung auch nicht erforderlich: Das Grundkonzept von XML besagt bereits, dass stets alle Marken zu setzen sind.

- **Prinzip 1:**
  In XHTML dürfen keine Marken entfallen. Ein Elementcontainer *muss* in XHTML stets *geschlossen* werden.

  Der Hintergrund: XML-Dokumente sollen *genau definierte Containerbereiche* besitzen und den Parser von der Erfordernis befreien, Marken (seien es Start- oder Endmarken) zu ergänzen. Start- und Endmarken sind also immer durch den Autor selbst zu setzen.

Unterschied 2 – Abschluss auch für leere Elemente

Die anderen, bereits für HTML vorgestellten Vereinbarungen, gleichen ihren Gegenstücken für XHTML ebenfalls weitgehend und müssen hier nicht vorgestellt werden. Ein Blick auf die Definition des Zeilenumbruchs lohnt allerdings (Abbildung 2.12).

Nicht, dass diese Vereinbarung großartig anders aussehen würde (das Inhaltsmodell `EMPTY` existiert auch in XML-Vereinbarungen). Als folgenreich erweist sich jedoch die Forderung, dass ein Element *stets geschlossen* werden muss: Es gibt keine Möglichkeit, in XML-Vereinbarungen festzulegen, dass für ein Element nur eine Startmarke existieren soll.

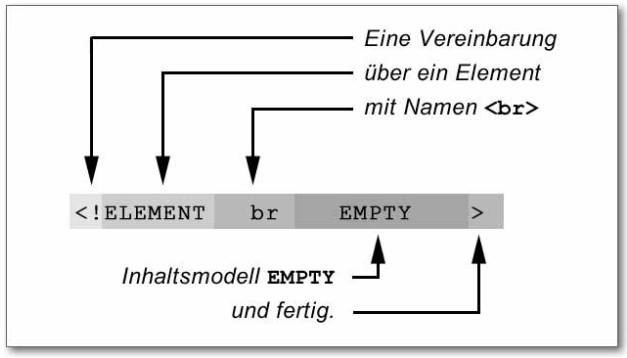

Abbildung 2.12 XML-konforme Vereinbarung des Zeilenumbruchelements

Ein XML-Parser würde wegen des fehlenden Elementabschlusses einen Fehler melden, wenn einfach `<br>` geschrieben würde. Aus diesem Grunde wurde in XML für **leere Elemente** ein **anderer closing delimiter** der Marke vereinbart, um eine Startmarke von einem leeren Element zuverlässig unterscheiden zu können (Abbildung 2.13). In XHTML ist die korrekte Schreibweise daher `<br/>`.

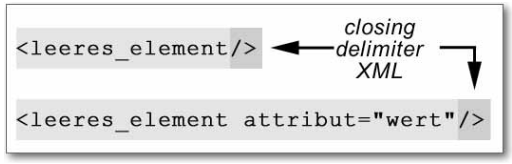

Abbildung 2.13 Anderer »closing delimiter« für leere Elemente in XML

- **Prinzip 2:**
  Leere Elemente werden in XHTML mit dem »closing delimiter« `/>` in der Startmarke *intern abgeschlossen*.

Elemente, die nicht als Container definiert sind, besitzen keine Endmarke, müssen also in der Form `<leeres_element/>` abgeschlossen werden. Nach dem Elementnamen und vor dem Abschluss dürfen jedoch Leerzeichen stehen. Korrekt ist also sowohl die Schreibweise `<br/>` als auch `<br />`.

Achtung – ein leeres Element ist kein Container (und umgekehrt)

Problematisch ist das (syntaktisch jedoch nicht falsche) Hinzusetzen einer Endmarke zu einem als »leer« definierten Element, also eine Form `<br></br>` , da es sich hier nicht wirklich um einen Container handelt. Hat man es allerdings mit einem (zufällig) leeren Container wie `<p></p>` zu tun, so ist wiederum eine verkürzte Schreibweise `<p/>` hierfür abzulehnen. Beide Fälle, »falsche« Container und »falsche« leere Elemente, führen gelegentlich zu Ausgabefehlern.

Unterschied 3 – Kleinschreibung für alle Bezeichner

Die Tatsache, dass in XHTML alle Bezeichner für Elemente und Attribute klein geschrieben werden müssen, ist anhand der Vereinbarungen nicht so augenfällig zu bemerken. Im Gegensatz zu SGML ist XML »case sensitive«, erachtet die Schreibweise der Namen also als relevant. Für XHTML musste, da `<P>` und `<p>` verschiedene Elemente kennzeichnen würde, eine Entscheidung getroffen werden, die zu Gunsten der **Kleinschreibung** ausfiel.

```
<P>Dies ist korrektes HTML, aber falsch in XHTML.</P>
<p>Dies ist korrektes HTML und korrektes XHTML.</p>
```

- **Prinzip 3:**
  Alle Element- und Attributnamen werden in Kleinbuchstaben geschrieben. Die Verwendung von Großbuchstaben macht das Dokument ungültig.

Unterschied 4 – Attributwerte stehen in Anführungszeichen

Es ist zwar in HTML stets korrekt, Attributwerte in Anführungszeichen einzuschließen (man sollte dies daher tun), man darf sie aber unter bestimmten Umständen weglassen. In XHTML ist es dagegen Pflicht, Attributwerte *immer* mit **Anführungszeichen** zu begrenzen. Dies können wahlweise einfache oder doppelte sein.

- **Prinzip 4:**
  Attributwerte stehen in Anführungszeichen.

Unterschied 5 – Ein Attribut muss einen Wert besitzen

Anhand der bisherigen Beispiele ist dies zwar nicht belegt, jedoch existieren in HTML, analog zu den leeren Elementen auch »leere« Attribute, deren bloßes Vorhandensein im Starttag ausreicht. Dies ist in XHTML nicht gestattet, da Attributen stets explizit ein Wert zugewiesen werden muss. Für die leeren Attribute aus HTML ist es Usus, ihnen als Wert ihren Attributnamen zu zuweisen:

Beispiel: Statt wie in HTML `<HR NOSHADE>`, steht in XHTML `<hr noshade="noshade"/>` **[zB]**

- **Prinzip 5:**
  Jedem Attribut muss ein Wert zugewiesen werden. **Leere Attribute** sind in XHTML *verboten*.

Unterschied 6 – Wurzelelement <html> vorgeschrieben

In HTML ist das Wurzelelement `<HTML>` optional und wird, wenn es fehlt durch den Parser ergänzt. In XHTML muss es gesetzt werden, da hier ein XML-Format

vorliegt, das ein Wurzelelement zwingend vorschreibt. Ebenso zwingend ist die Schreibweise als `<html>`.

- **Prinzip *6*:**
  Es muss ein Wurzelelement mit Namen `<html>` vorhanden sein.

Unterschied 7 – Es muss ein XHTML-Namensraum vergeben sein

Elementbezeichner alleine genügen bei XML nicht, um die **qualitative Zugehörigkeit** eines Elements zum Vokabular einer bestimmten **Markup-Sprache** zu definieren: Immerhin können Bezeichner ja (im Prinzip) beliebig gewählt und XML-Dokumenttypen obendrein gemischt werden.

XML kennt zur Lösung so genannte »Namensräume«, die an die Elemente »gebunden« werden. Dies kann man sich wie eine zusätzliche (und auslesbare) Eigenschaft vorstellen, die jedes Element erhält. Der Elementbezeichner wird dadurch »qualifiziert«, d.h. die Zugehörigkeit zu einer bestimmten Sprache ist eindeutig bestimmt. Ein **Namensraum** ist dabei nichts weiter als ein unsichtbarer zweiter Bezeichner, den ein Element (oder Attribut) zusätzlich zu seinem eigenen Namen (dem »lokalen Bezeichner«) erhält.

Die zur Kennzeichnung eines Namensraumes eingesetzte Zeichenkette muss jedoch eindeutig sein. Damit Dopplungen vermieden werden, verwendet man als Bezeichner einen **URI-String**, der per Definition eindeutig und unverwechselbar ist. Als Verweis auf eine Ressource (gleich einem Hyperlink) ist ein Namensraum-URI jedoch nicht zu verstehen.

Für XHTML lautet der **Namensraumstring** `http://www.w3.org/1999/xhtml`. Er wird dem Wurzelelement `<html>` mittels des Attributs `xmlns` zugewiesen und kennzeichnet das Dokument und seine Elemente eindeutig als XHTML:

Abbildung 2.14 Der Namensraum kennzeichnet ein Vokabular eindeutig.

Praktischerweise werden damit gleich **alle restlichen Elemente** des Dokuments mit erfasst: Der dem Wurzelelement zugewiesene Namensraum gilt für das Wurzelelement selbst und alle seine Abkömmlinge. Den anderen Elementen muss der Namensraum nicht extra zugewiesen werden – aus diesem Grund nennt man den über das `xmlns`-Attribut zugewiesenen Namensraum auch den **Default-Namensraum**.

- **Prinzip 7:**
  Ein XHTML-Dokument **muss** sich im XHTML-Namensraum befinden.

Unterschied 8 – Eine Dokumenttyp-Deklaration muss vorhanden sein

Damit ein XHTML den formalen Regeln vollkommen genügt, muss auch sein Dokumenttyp festgelegt werden. Dies geschieht in Form einer so genannten **Dokumenttyp-Deklaration** (»doctype declaration«; nicht zu verwechseln mit der »doctype definition«, in der das Sprachvokabular beschrieben wird). Diese Deklaration hat die Aufgabe, die dem Dokument zugrunde liegende DTD zu bezeichnen. Sie wird dem Wurzelelement vorangestellt (Abbildung 2.15).

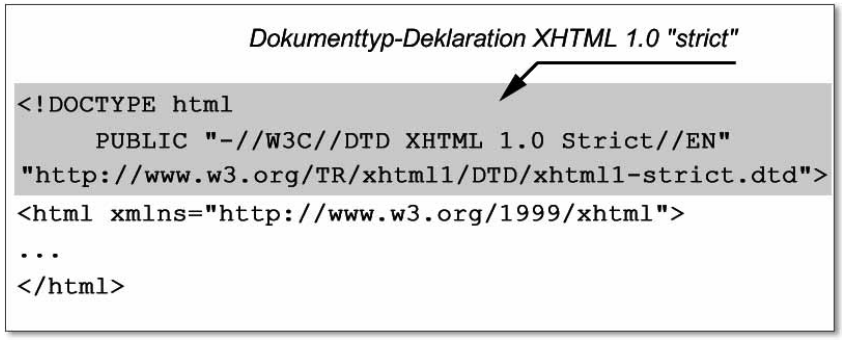

Abbildung 2.15 XHTML-Dokument mit Dokumenttyp XHTML 1.0 »strict«

XHTML 1.0 existiert in den drei DTD-Varianten »strict«, »transitional« und »frameset«, die folgendermaßen lauten:

```
<!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">
```

Die Wahl einer DTD-Variante bestimmt das für das jeweilige Dokument erlaubte Sprachvokabular, an die sich das Dokument dann auch zu halten hat. Die liberale Variante ist »transitional«, die orthodoxe ist »strict«. Die Variante »frameset« wird nur für Frameset-Dokumente benötigt.

- **Prinzip 8:**
  Ein XHTML-Dokument *muss* eine **Doctype-Deklaration** besitzen.

2.3.1 Das Vokabular von XHTML

Der Sprachumfang von *XHTML 1.0 frameset* entspricht vollständig dem von *HTML 4.01 frameset*. Auch hier existieren dieselben 91 Elemente. Im Unterschied zu HTML müssen die Bezeichner jedoch klein geschrieben werden.

<a> <abbr> <acronym> <address> <applet> <area />
<b> <base /> <basefont /> <bdo> <big> <blockquote> <body>   <button>
<caption> <cite> <code> <col> <colgroup>
<dd> <del> <dfn> <dir> <dl> <dt>
<fieldset> <font> <form> <frame /> <frameset>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr> <html>
<i> <iframe> <img /> <input /> <ins> <isindex />
<kbd> <label> <legend>   <link>
<map> <menu> <meta />
<noframes> <noscript>
<object>   <optgroup> <option>
<p> <param /> <pre> <q>
<s> <samp> <script> <select> <small> <strike> <style> <sub> <sup>
<table> <tbody> <td> <textarea> <tfoot> <th> <thead> <title> <tr> <tt>
<u>   <var>

Tabelle 2.2 Elemente von XHTML 1.0 frameset

2.4 Zusammenfassung

Aus diesem Kapitel sollten Sie folgende Kernaussagen mitnehmen:

- HTML und XHTML haben **dieselbe Aufgabe** und gleichen sich daher in vielen Belangen, z. B. bei der Anzahl und Bezeichnung der Elemente und Attribute und deren Verwendung.
- Da HTML und XHTML **unterschiedlicher Herkunft** sind, gibt es prinzipbedingt auch Unterschiede zwischen beiden Sprachen. Diese sind in der folgenden Tabelle kurz zusammengefasst.

	HTML	XHTML
Wurzelelement	optional	obligatorisch
Namensraum	Namensraumkonzept existiert nicht	XHTML-Namensraum erforderlich
Doctype-Deklaration	optional	erforderlich
Elementverkürzungen	gestattet	nicht gestattet
Attributwerte	Anführungszeichen um Zahlenwerte optional	Anführungszeichen obligatorisch
Marker-Attribute	gestattet	nicht gestattet
Leere Elemente	werden nicht geschlossen	müssen geschlossen werden
Bezeichner	Schreibweise nicht reglementiert	Kleinschreibung vorgeschrieben

Tabelle 2.3 Zusammenfassung der Unterschiede zwischen HTML und XHTML

- Leere SGML-Vereinbarungen der Form `<!-- Kommentar -->` dienen in HTML und XHTML als **Kommentare**. Sie können überall im Quelltext von Dokumenten eingesetzt werden.

Wie geht es weiter?

Das folgende Kapitel erklärt den allgemeinen Aufbau von HTML- und XHTML-Dokumenten und erläutert Aufbau und Aufgaben von Dokumentkopf und -rumpf. Es folgt eine nähere Betrachtung der Dokumenttyp-Deklaration und eine exemplarische Vorstellung des Dokumentaufbaus aller aktuellen HTML- und XHTML-Dokumenttypen.

Welche der Spielarten von HTML soll man als Webautor wählen und was ist die Grundstruktur der Dokumente? Wie sind Dokumentkopf und Dokumentrumpf aufgebaut und welche Elemente sind dort jeweils erlaubt? Wie funktioniert eine Dokumenttyp-Deklaration? Dieses Kapitel gibt Antwort.

3 (X)HTML – die Dokumentstruktur

Derzeit gültig sind, wie im ersten Kapitel dargelegt wurde, sieben verschiedene *aktuelle* Varianten von HTML – diejenigen für Mobilgeräte nicht eingerechnet. Im Prinzip könnten sogar die älteren Versionen HTML 2.0 und HTML 3.2 noch eingesetzt werden (sofern man deren Defizite in Kauf nimmt, was sicher niemand will), da aktuelle Browser auch solche Seiten darstellen.

3.1 Die Wahl der geeigneten Sprachversion

Bleibt die Grundsatzentscheidung, ob man nun HTML verwenden soll – oder besser XHTML. Eine solche **Entscheidung** muss jedoch getroffen werden: Mischformen wie die Anreicherung von HTML-Dokumenten mit XHTML-Syntax (z. B. durch das Schließen leerer Tags) sind nicht etwa eine »Verbesserung«, sondern schlichtweg falsch.

Tatsache ist, dass XHTML »moderner« ist. Soll man es deshalb vorziehen? Tatsache ist auch, das XHTML in vielen Fällen genauso behandelt wird wie HTML. Ist der Schritt von HTML zu XHTML dann nicht überflüssig? Drei Optionen lassen sich unterscheiden, die aber stets auf dem gleichen Fundament aufsetzen sollten: »gut geschriebenem« Quellcode.

3.1.1 Möglichkeit 1: Korrekt geschriebenes HTML

Webserver liefern HTML wie XHTML gewöhnlich mit dem **MIME-Typ** `"text/html"` aus, behandeln sie also identisch – der XML-Aspekt von XHTML kommt nicht zum Tragen. In diesem Falle könne man ebenso gut HTML einsetzen, mag man einwenden. Im Prinzip ist das richtig: **Diszipliniert geschriebenes HTML** kann eine adäquate Lösung darstellen, wenn es um Information geht, die primär durch Webbrowser ausgewertet wird.

Das »disziplinierte« Schreiben bedeutet, auf alle (eigentlich erlaubten) Verkürzungen zu verzichten und Präsentationselemente außen vor zu lassen. Es läuft also kurz gesagt darauf hinaus, sich unter **Beachtung der HTML-Syntax** soweit wie möglich an XHTML anzunähern. Die Variante HTML 4.01 »strict« ist hervorragend auf die Zusammenarbeit mit CSS eingerichtet, auf die es hier im Wesentlichen ankommt. Auch die »transitional«-Version kommt in Frage, da sie Freiheiten bietet (Stichwort: `target`-Attribut), die in »strict« nicht zur Verfügung stehen (es gibt ja keinen *Zwang*, die dann erlaubten Präsentationselemente und -attribute zu verwenden).

Sollte man in die Lage kommen, **nicht spezifizierte Elemente oder Attribute** (also proprietäre, durch Browserhersteller eingeführte) verwenden zu *müssen* – dies kann in einem Intranet durchaus wünschenswert sein, wenn eine Kontrolle über die installierten Zielbrowser vorhanden ist –, so wird HTML wiederum noch interessanter: eine Dokumenttyp-Deklaration ist ja optional, weshalb gefahrlos gegen diese Vorschriften verstoßen werden kann. Generell werden HTML-Dokumente eher selten validiert, also auf Gültigkeit gegenüber einem Dokumenttyp geprüft.

Was versteht man unter einem MIME-Typ?

MIME ist das Akronym für **Multipurpose Internet Mail Extensions**, stammt also aus dem E-Mail-Bereich und der dort aufgetretenen Erfordernis, den Inhaltstyp, also die »Art« einer Datei zu bezeichnen. Hierbei wird stets ein zweiteiliger Bezeichner verwendet, der folgende Form besitzt:

```
haupttyp/untertyp
```

So bezeichnet `"text/html"` ein Textformat in der Ausprägung »html«, der Typ `"application/xhtml+xml"` ein Dokument in der Ausprägung »xhtml«, der für eine Anwendung (application) bestimmt ist, die XML verarbeitet.

Da die Bezeichnungen standardisiert sind, werden die gleichen MIME-Typen auch in Zusammenhang mit dem Hypertext Transfer Protocol (HTTP) eingesetzt und kennzeichnet dort eine durch den Webserver versendete Datei. In HTML werden MIME-Typbezeichner verwendet, um Datentypen von Containern oder für den Formularversand zu benennen.

MIME-Standard RFC 2045: http://tools.ietf.org/html/rfc2045

3.1.2 Möglichkeit 2: Wohlgeformtes XHTML

Ein HTML-Dokument mag noch so korrekt und vollständig geschrieben sein, es ist dadurch noch lange kein **XML-konformes Dokument**. Der Vorteil von XML liegt ja in der Möglichkeit, die Information auch anders als durch Browser auszuwerten. Die regelmäßige und verlässliche Struktur erlaubt die Eingliederung von

Webdokumenten in XML-Informationsstrukturen, die diese beispielsweise mit XSLT, XQuery oder vergleichbaren (vielleicht noch gar nicht erfundenen) Techniken »anzapfen« können. Kurz, XHTML ist die zukunftsträchtigere Option, wenn auf langfristige Sicht geplant werden muss. XHTML 1.0 »strict« oder »transitional« sind die Technologien der Wahl, wenn Dokumente auch außerhalb des `"text/html"`-Kontexts genutzt werden (sollen).

Unter »wohlgeformt« versteht sich die **Befolgung der XHTML-Syntax**, wie sie im vorangegangenen Kapitel erläutert wurde. Sofern diese Regeln befolgt werden, mag auch der Einsatz proprietärer Elemente und Attribute (natürlich in XHTML-Schreibweise) verzeihlich sein. Allerdings degradiert dies die obligatorische Dokumenttyp-Deklaration zur bloßen Behauptung: Validieren lässt sich so ein Dokument nicht in jedem Fall, wohl aber erfolgreich in einer XML-Umgebung verwenden.

3.1.3 Möglichkeit 3: Streng konformes XHTML

Manchmal ist die Befolgung eines Dokumenttyps nicht nur als Lippenbekenntnis, sondern in der Realität unabdingbar. Wenn absolute Zuverlässigkeit und Konformität der Informationsstruktur gefragt sind, muss das Dokument eine **Validierung** durchlaufen. Sobald die Voraussetzungen für ein korrektes XHTML-Dokument nicht nur formal, sondern auch tatsächlich erfüllt sind, haben Sie ein **streng konformes XHTML-Dokument** vor sich. Ob dies einen Wert an sich darstellt, sei dahingestellt. In jedem Fall bedeutet es einen zusätzlichen Arbeitsschritt (und gegebenenfalls nachfolgend Korrekturen), aber auch die Gewissheit, ein verwendbares Dokument erstellt zu haben.

Sollten Sie XHTML-Dokumente automatisch generieren und dokumentierend in Geschäftsvorgänge einfließen lassen wollen, so ist eine Validierung sicher keine schlechte Idee, da die Auswertbarkeit der Information dann sichergestellt ist (natürlich nur, sofern die verwendeten Daten korrekt sind).

3.1.4 Ein paar Überlegungen zu XHTML 1.1

Die Unterschiede zwischen XHTML 1.0 und XHTML 1.1 sind größer, als auf den ersten Blick zu bemerken. Zwar sind Elemente und Syntax praktisch identisch mit denen von XHTML 1.0 »strict«, eine der Randbedingungen richtet jedoch gewaltige Hürden auf: **XHTML 1.1 muss als XML behandelt werden**.

Dies bedeutet, dass es mit der »bequemen« Auslieferung mit dem MIME-Typ `"text/html"` vorbei ist – für XHTML 1.1 ist `"application/xhtml+xml"` verbindlich. Die Konsequenz ist, dass auch der Browser ein solches Dokument als XML zu behandeln hat, es also im Falle von Syntaxverstößen zurückweist. Zu diskutie-

ren wäre, ob dies dem »liberalen« Geiste des Internets und der eigentlich beabsichtigten Plattformunabhängigkeit von HTML noch entspricht.

Wenn Sie einige Funktionen wie Ruby-Elemente benötigen, die nur XHTML 1.1 bietet, *und* Ihre Dokumente ohnehin validieren *und* die Auslieferung als XML kein Problem darstellt, *dann* können Sie XHTML 1.1 in Betracht ziehen.

Was versteht man unter wohlgeformt, typgültig und Validierung?

Diese drei Begriffe stammen aus XML. Der Ausdruck **Wohlgeformtheit** bezeichnet die Korrektheit der Syntax eines XML-Dokuments, also das Vorhandensein eines Wurzelelements, die Befolgung der Verschachtelungsregeln und der Schreibweise der Bezeichner. Es ist in diesem Falle *nicht* Pflicht, dass eine Dokumenttyp-Deklaration vorliegt.

Als Erweiterung der Wohlgeformtheit kann die **Typgültigkeit** betrachtet werden: Ein Dokument ist wohlgeformt (*well-formed*) und besitzt eine Dokumenttyp-Deklaration, der das Dokument auch entspricht (*valid*). Streng genommen muss dies für XHTML-Dokumente immer gelten.

Unter **Validierung** versteht man die Prüfung eines Dokuments auf Typgültigkeit durch eine Software, den so genannten Validator. Dieser ist in der Lage, Fehler im Dokument zu erkennen und anzuzeigen.

3.2 Grundaufbau von HTML- und XHTML-Dokumenten

Der Grundaufbau von HTML- und XHTML-Dokumenten ist, abgesehen von sprachspezifischen Details, identisch (siehe Abbildung 3.1 und Abbildung 3.2). Im **Wurzelelement** `<html>` befinden sich die beiden funktionalen Dokumentbereiche `<head>` als **Dokumentkopf** und `<body>` als **Dokumentrumpf**. Ersterer stellt einen Container für den Dokumenttitel `<title>` und optional weitere dokumentbezügliche Informationen dar, Letzterer enthält alle sichtbaren Bestandteile des Dokuments. Für die Grundstruktur des Dokuments werden also nur vier Elemente benötigt, bei denen es sich jeweils um Container handelt. Man bezeichnet sie als **Strukturelemente**.

Wie in Abbildung 3.1 angedeutet, kann auch ein HTML-Dokument eine **Dokumenttyp-Deklaration** besitzen. Da dies optional ist, wurde sie in der Abbildung ausgegraut. Beachten Sie, dass die Typdeklaration nur einen **Prolog** zum Dokument darstellt und nicht zu dessen Inhalt zählt: Das eigentliche Dokument besteht aus dem Wurzelelement und seinem Inhalt.

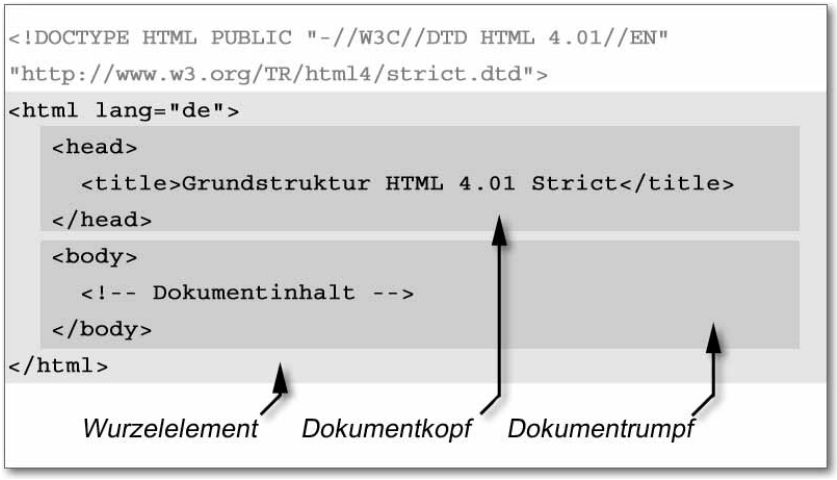

Abbildung 3.1 Grundaufbau eines HTML-Dokuments

Ebenfalls nicht Pflicht ist das `lang`-Attribut, das hier für das Wurzelelement gesetzt wurde (es könnte als so genanntes »Universalattribut« prinzipiell jedem der hier vertretenen Elemente zugewiesen werden). An dieser Position gilt es für den gesamten Dokumentinhalt und bezeichnet diesen als »in deutscher Sprache geschrieben«. Eine solche Sprachangabe erleichtert die Katalogisierung der Seite und ihre Interpretation (beispielsweise durch Bildschirmlesegeräte, die ihre Ausspracheregeln danach einrichten können) – sie ist also im Sinne der Barrierefreiheit von Vorteil.

Was versteht man unter Barrierefreiheit?

Mit »Barrierefreiheit« bezeichnet man den Umstand, dass die Informationen einer Webseite für möglichst **alle Nutzergruppen** zugänglich sind und dies **ohne Einschränkung**. Dies ungeachtet technischer Limitierungen (der User Agents) oder physischer Handicaps (auf Seiten der Nutzer) zu ermöglichen, erfordert Maßnahmen bereits auf Quelltextebene, aber auch im Seitenlayout selbst.

Das XHTML-Dokument, das schematisch in Abbildung 3.2 gezeigt wird, verfügt ebenfalls über einen Prolog in Form einer **Dokumenttyp-Deklaration**, der hier allerdings **obligatorisch** ist. Ebenso obligatorisch ist die Vergabe des **XHTML-Namensraums**. Anstelle des `lang`-Attributs sehen Sie hier das `xml:lang`-Attribut, das dessen Aufgabe für XML-Dokumente erfüllt. Der restliche Aufbau aus Dokumentkopf und -rumpf ist identisch.

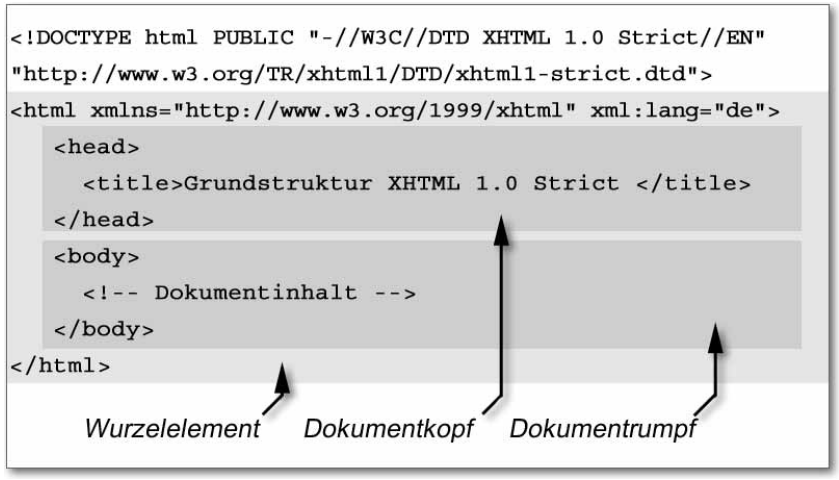

Abbildung 3.2 Grundaufbau eines XHTML-Dokuments

[»] *Hinweis:* Bitte beachten Sie den Einsatz des **HTML-Kommentars** innerhalb des `<body>`-Containers, der stellvertretend für dessen Inhalt steht.

Hier folgt als Übersicht eine tabellarische Aufstellung der vier Strukturelemente:

Element	Rolle	Typ
`<html> ... </html>`	Wurzelelement des Dokuments	Strukturcontainer
`<head> ... </head>`	Dokument*kopf*; enthält neben dem Titel weitere Metainformationen	Strukturcontainer
`<title> ... </title>`	Dokument*titel*; Teil des Dokumentkopfes	Strukturcontainer
`<body> ... </body>`	Dokument*körper*; enthält den sichtbaren Teil des HTML-Dokuments	Strukturcontainer

Tabelle 3.1 Grundelemente eines HTML-Dokuments

Strukturelemente sind Pflichtelemente

Jedes der vier Strukturelemente `<html>`, `<head>`, `<body>` und `<title>` muss in einem Dokument des Typs »strict« oder »transitional« genau einmal und an der richtigen Position erscheinen. Für XHTML ist dies vorgeschrieben. Für gut geschriebene HTML-Dokumente sollte die Regel ebenfalls befolgt werden.

Das Frameset-Dokument – Der »dritte« Dokumenttyp

Ohne dieses Thema vertiefen zu wollen, möchte ich an dieser Stelle kurz eine Erläuterung zu **Frameset-Dokumenten** einschieben. Dass eine weitere DTD exis-

tiert, die die Elemente zur Erstellung solcher Dokumente enthält, ist bekannt. An dieser Stelle muss angemerkt werden, dass sich Frameset-Dokumente von den oben gezeigten Grundtypen darin unterscheiden, dass sie selbst keine sichtbare Oberfläche besitzen, daher auch keinen Dokumentkörper `<body>`. An dessen Stelle tritt der `<frameset>`-Container (Abbildung 3.3).

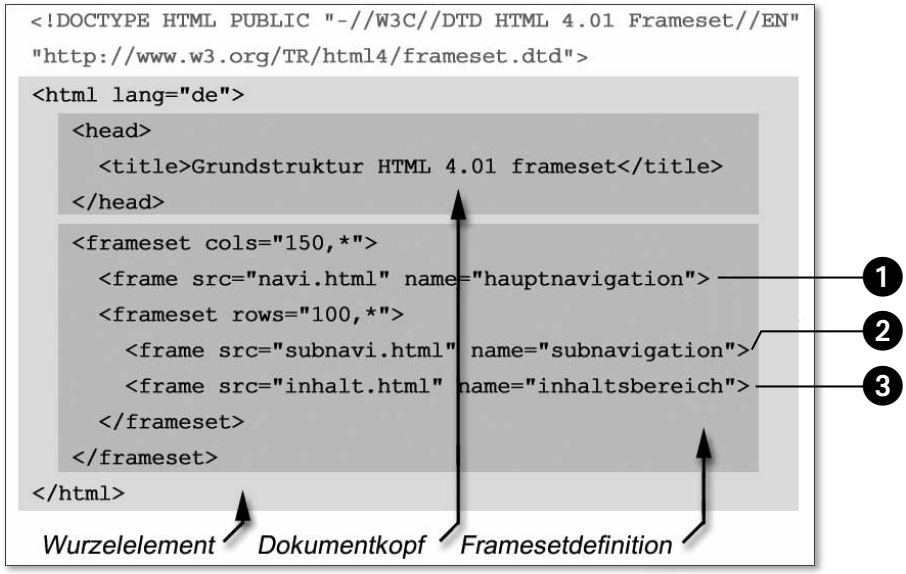

Abbildung 3.3 Grundaufbau eines HTML-Frameset-Dokuments

Andere Strukturelemente im Frameset-Dokument

Im Frameset-Dokument tritt der `<frameset>`-Container an Stelle von `<body>` auf. Die Elemente `<html>`, `<head>` und `<title>` hingegen müssen ebenso verwendet werden wie in den anderen Dokumenttypen. Das Element `<body>` ist ebenfalls zugelassen, aber nur innerhalb des optionalen Containers `<noframes>`, der Alternativinhalt zum Frameset enthält.

Der optionale Prolog mit der Dokumenttyp-Deklaration (hier natürlich eine für die *»frameset«*-DTD), das Wurzelelement und der Dokumentkopf sind identisch mit denen des soeben vorgestellten HTML-Dokuments. Das Strukturelement `<frameset>` enthält die `<frame>`-Definitionen, die auf weitere HTML-Dokumente verweisen. Diese ergeben dann die Oberfläche des Framesets (korrespondierende Ziffern ❶–❸). Das Frameset-Dokument sorgt lediglich für die Aufteilung der Bildschirmfläche (siehe Abbildung 3.4), in der die eingebundenen Dokumente gezeigt werden.

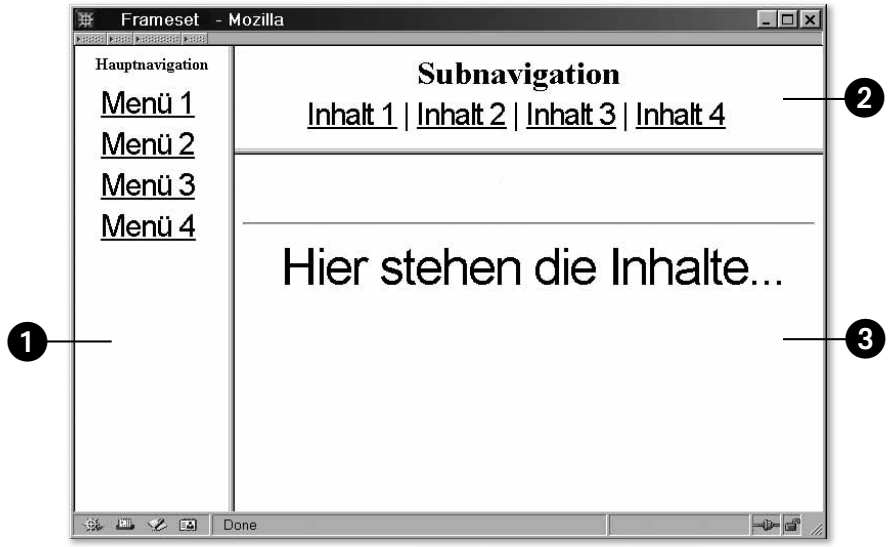

Abbildung 3.4 Präsentation eines Frameset-Dokuments im Browser

Element	Rolle	Typ
`<frameset> ... </frameset>`	Gruppiert Frame- und Framesetdefinitionen zu einem Frameset	Strukturcontainer
`<frame />`	Definiert einen Frame innerhalb eines Framesets	Leeres Element

Tabelle 3.2 Die zusätzlichen Grundelemente eines Frameset-Dokuments

Framesets – Sollte man sie einsetzen? Oder besser nicht?

Framesets gelten als nicht mehr zeitgemäß. Die durch sie zu erzielende Aufteilung des Bildschirms in Bereiche mit stationären oder wechselnden Inhalten kann heutzutage auch mit CSS-Anweisungen nachgebildet werden. Zudem stellen Framesets eine Barrierehürde dar, die nicht von allen User Agents (Suchmaschinen, Mobilgeräte) genommen werden kann. Eine Diskussion dieser Problematiken finden Sie im Referenzteil zu den Elementen.

Dennoch haben Frames eine Daseinsberechtigung für bestimmte Anwendungen (Dokumente aus verteilten Quellen), weshalb das W3C auch an **XFrames**, einer zeitgemäßen Neuformulierung arbeitet (**www.w3.org/TR/xframes**).

3.3 Aufbau und Aufgaben des Dokumentkopfes

Von der Vielzahl möglicher Inhalte des Dokumentkopfes wurde bislang nur der <title>-Container vorgestellt, der als obligatorisches Element auch zu den strukturellen Bestandteilen eines vollständigen HTML-Dokuments gehört.

Es existieren allerdings weitere optionale Elemente, die innerhalb des <head>-Containers erscheinen dürfen. Ihre Anzahl und Reihenfolge sind nicht reglementiert. In Abbildung 3.5 sind Beispiele für die erlaubten Inhalte des Dokumentkopfes und ihre Aufgaben gezeigt. Die Dokumenttyp-Deklaration wurde der Übersichtlichkeit halber weggelassen.

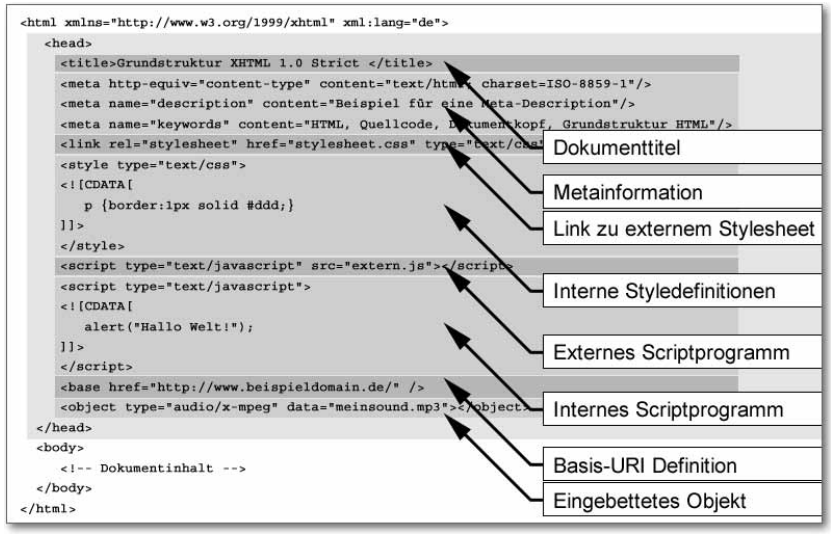

Abbildung 3.5 Die im <head>-Container erlaubten HTML-Elemente

Hierbei handelt es sich teilweise um **leere Elemente**, teilweise um **Container**. Zwei von den letzteren, der <script>- und der <style>-Container, besitzen in HTML und XHTML ärgerlicherweise verschiedene Inhaltsmodelle, so dass sie mit unterschiedlicher Syntax eingesetzt werden müssen.

In der Übersicht sind, nach <title>, der Reihe nach eingebunden:

Element	Rolle	Typ
<meta ... />	Metainformationen zum Dokument wie Beschreibung, Keywords, Zeichensatz	Leeres Element; Headkontext

Tabelle 3.3 Übersicht über die im Dokumentkopf erlaubten Elemente

Element	Rolle	Typ
`<link ... />`	Bindet eine externe Ressource (Stylesheet o.ä.) ein	Leeres Element; Headkontext

`<style>`
`  // CSS-Anweisungen`
`</style>`
	Enthält CSS-Deklarationen oder die Importanweisung eines externen Stylesheets	Container (Dokumentkopf)

`<script src="extern.js">` `</script>`	Bindet externes Script als Datei ein	Leerer (!) Container

`<script>`
`  // Scriptbefehle`
`</script>`
	Scriptcontainer für interne Scriptanweisungen	Container

`<base href="..." />`	Definiert den Basis-URI des Dokuments	Leeres Element
`<object> ... </object>`	Fügt ein Objekt ein	Container

Tabelle 3.3 Übersicht über die im Dokumentkopf erlaubten Elemente (Forts.)

Metainformation mit <meta />

Das `<meta>`-Element fügt dem Dokument **Metainformationen** wie Beschreibung, Schlagworte, Autor oder Zeichenkodierung hinzu. Im Beispiel sind drei Metaelemente verwendet. Das erste nennt den **Inhaltstyp** des Dokuments, also seinen Zeichensatz und seinen MIME-Typ. Da diese Information »äquivalent« zu normalerweise innerhalb des HTTP-Headers durch den Webserver gegebenen Information ist, wird der Bezeichner `"content-type"` dieser Metainformation mit dem Attribut `http-equiv` festgelegt. Der eigentliche Informationsinhalt liegt bei `<meta/>`-Tags immer als Wert des `content`-Attributs vor.

```
<meta http-equiv="content-type"
 content="text/html; charset=ISO-8859-1"/>
```

Die folgenden Metainformationen sind keine HTTP-Äquivalente, daher erfolgt ihre Bezeichnung mit dem `name`-Attribut. Ihr Inhalt steht wieder im `content`-Attribut. Es handelt sich um eine **inhaltliche Beschreibung** des Dokuments (`"description"`) und **Schlüsselwörter** (`"keywords"`); beides sind Informationen, die sich primär an Suchmaschinen richten.

```
<meta name="description"
 content="Beispiel für eine Meta-Description"/>

<meta name="keywords"
 content="HTML, Quellcode, Dokumentkopf, Grundstruktur"/>
```

Welche Metainformationen sind erforderlich?

Erforderlich im Sinne des Wortes sind keine dieser Informationen. Als *sinnvoll* betrachtet werden jedoch die an Suchmaschinen gerichteten Informationen über Dokumentinhalt, Verschlagwortung, die Informationen über Dokumenttyp und gegebenenfalls die im Dokument eingesetzte Scriptsprache.

Hinweis: Mehr über das `<meta>`-Element erfahren Sie in Abschnitt 14.55, *Elementreferenz*, <*meta*>. **[«]**

Externe CSS-Stylesheetdatei mit <link />

Das leere Element `<link>` tritt ausschließlich im `<head>`-Kontext auf. Es dient zur Definition einer Verknüpfung des aktuellen Dokuments mit einer externen Ressource, wobei mit `rel`- oder `rev`-Attribut die Art der Beziehung zwischen Dokument und Ressource beschrieben werden kann.

Die gebräuchlichste Verwendung von `<link>` ist das Einbinden externer Stylesheetdateien, hierfür muss `rel="stylesheet"` gesetzt sein:

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

Hinweis: Mehr über das `<link>`-Element erfahren Sie in Abschnitt 14.50, *Elementreferenz*, <*link*>. **[«]**

Dokumentstyle-Container <style>

Der `<style>`-Container dient zum Einfügen von Stylesheetanweisungen in den Dokumentkopf. Sie gelten für das gesamte Dokument und werden daher als **Dokumentstyle** bezeichnet. Obligatorisch ist die Angabe des MIME-Typs mittels des `type`-Attributs. Normalerweise ist dessen Wert `"text/css"`, da dies die einzige gebräuchliche Stylesprache ist.

Das **Inhaltsmodell** des Containers ist »`CDATA`-Textinhalt« für HTML- und »`PCDATA`-Textinhalt« für XHTML-Dokumente. Um nicht womöglich die Styleanweisungen selbst anpassen zu müssen, um das abweichende Parserverhalten zu berücksichtigen, schließt man bei XHTML-Dokumenten den Containerinhalt in einen `CDATA`-Bereich ein. Hieraus resultieren unglücklicherweise auch wieder Kompatibilitätsprobleme, weshalb es ratsam ist, stattdessen auf externe, verlinkte Styleanweisungen zurückzugreifen.

Einbindung in HTML	Einbindung in XHTML

`<style type="text/css">`
`<!--`
`  p {font-family:arial}`
`-->`
`</style>`
	`<style type="text/css">`
	`<![CDATA[`
	`  p {font-family:arial}`
	`]>`
	`</style>`

Tabelle 3.4 Unterschiede in der Einbindung von CSS-Daten bei HTML und XHTML

[»] *Hinweis:* Mehr über den `<style>`-Container erfahren Sie in Abschnitt 14.76, *Elementreferenz, <style>*.

Scriptcontainer <script>

Das `src`-Attribut nennt den URI einer externen Datei, welche die in Zusammenhang mit dem `<script>`-Container auszuführenden Scriptdaten enthält. Die externen Scriptdaten verhalten sich, als ob sie direkt im Container enthalten wären. Der `<script>`-Container muss geschlossen werden, wobei, obwohl er leer bleibt, keine Verkürzung zu `<script ... />` gestattet ist. Das Pflichtattribut `type` nennt den MIME-Typ der verwendeten Scriptsprache.

```
<script type="text/javascript" src="extern.js"></script>
```

Der `<script>`-Container wird auch zum direkten Einbetten von **Scriptanweisungen** in den Dokumentkopf, aber auch beliebigen Positionen im Dokumentrumpf verwendet. Obligatorisch ist wieder die Angabe des MIME-Typs mittels des `type`-Attributs. Der meistverwendete Wert ist `"text/javascript"` für JavaScript-Daten, es können aber auch beliebige andere Scriptsprachen eingebettet werden, solange der Browser in der Lage ist, diese auszuführen.

Auch für diesen Container existieren die bei `<style>` bereits angesprochenen Unterschiede im Inhaltsmodell und entsprechende Kompatibilitätsprobleme, die es nahelegen, in XHTML die Einbindung externer Scriptdateien vorzuziehen.

Einbindung in HTML	Einbindung in XHTML

`<script type="text/javascript">`
`<!--`
`  alert("Hallo Welt!");`
`//-->`
`</script>`
	`<script type="text/javascript">`
	`<![CDATA[`
	`  alert("Hallo Welt!");`
	`]]>`
	`</script>`

Tabelle 3.5 Unterschiede in der Einbindung von Script-Daten bei HTML und XHTML

[»] *Hinweis:* Mehr über den `<script>`-Container erfahren Sie in Abschnitt 14.70, *Elementreferenz, <script>*.

Basis-URI Definition mit <base />

Das Element <base> legt das Basis-Target aller im Dokument auftretenden Hyperlinks sowie den Basis-URL aller relativen URLs fest. Das href-Attribut von <base/> bezeichnet mit einem absoluten URI-String den absoluten Basis-URI, auf den sich relative URIs im Dokument beziehen.

```
<base href="http://www.beispieldomain.de/" />
```

Hinweis: Mehr über das <base>-Element erfahren Sie in Abschnitt 14.9, *Elementreferenz, <base>*. **[«]**

Objekt mit <object>

Das <object>-Element dient zum Einbetten eines beliebigen Datenobjekts (z. B. Video, Audio, Flash, Java-Applet) in ein HTML-Dokument, normalerweise allerdings im <body>-Container. Im Dokumentkopf dürfen <object>-Container jedoch auch auftreten – allerdings muss feststehen, dass sie in diesem Fall keinen visuellen Output erzeugen und keinen darzustellenden Inhalt besitzen (Elemente für Übergabeparameter <param /> sind gestattet).

Auf diesem Wege kann beispielsweise das Einbinden eines Soundfiles für Hintergrundmusik erfolgen:

```
<object type="audio/x-mpeg" data="meinsound.mp3"></object>
```

Hinweis: Mehr über den <object>-Container erfahren Sie in Abschnitt 14.59, *Elementreferenz, <object>*. **[«]**

3.4 Aufbau und Aufgaben des Dokumentkörpers

Der Dokumentkörper, mit anderen Worten das <body>-Element, enthält den im Browserfenster sichtbaren Teil des Dokuments. Die Beschreibung der **Gestaltung** dieses Inhaltes wird die folgenden Kapitel in Anspruch nehmen, weshalb der Aufbau an dieser Stelle nur kurz angerissen werden soll.

Zwischen HTML und XHTML besteht ein kleiner **Unterschied im Inhaltsmodell** von <body>: Während in HTML dort die unmittelbare Einbettung von Text gestattet ist, darf dies in XHTML nicht geschehen. Dort sind als unmittelbarer Inhalt von <body> ausdrücklich *nur Blockelemente* zulässig.

Nur in HTML erlaubt	In HTML und in XHTML erlaubt

`<body>`
`<h1>Eine Überschrift.</h1>`
`Text unmittelbar in Body.`
`...`
`</body>`
	`<body>`
	`<h1>Eine Überschrift.</h1>`
	`<p>Text in Textabsatz.</p>`
	`...`
	`</body>`

Im Sinne einer möglichst großen Angleichung von HTML an XHTML empfehle ich, auch in HTML-Dokumenten keine Textdaten in `<body>` einzufügen.

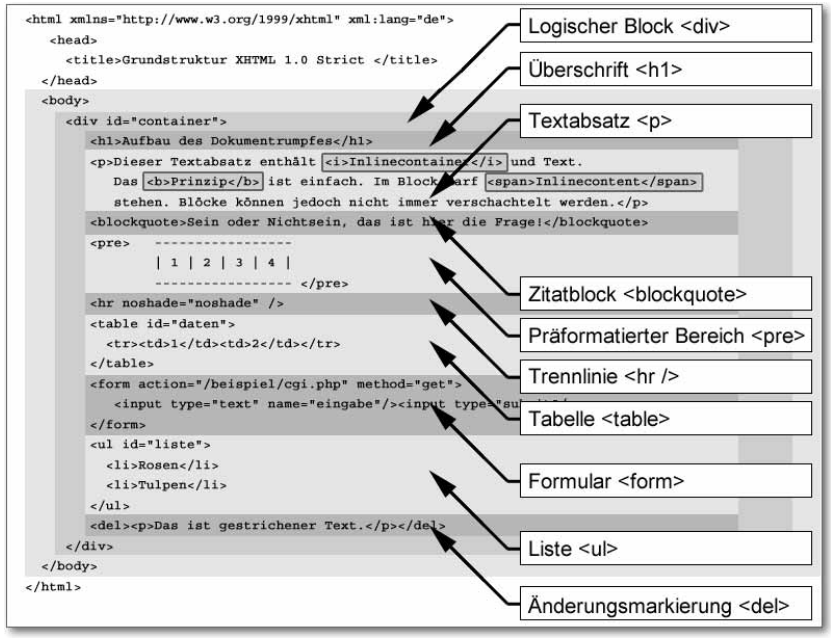

Abbildung 3.6 Schema erlaubter Elemente im Dokumentrumpf <body>

Logischer Block

Im Beispieldiagramm ist der gesamte Inhalt von `<body>` von einem **logischen Block** umschlossen, einem `<div>`-Container mit Identifier mittels `id`-Attribut. Dieser Identifier kennzeichnet den Container eindeutig, da er nur ein einziges Mal mit diesem Wert verwendet werden darf:

```
<div id="container">
  ...
  <!-- alle weiteren Inhalte -->
  ...
</div>
```

Dies ermöglicht es (wie Sie demnächst sehen werden), mittels CSS-Anweisungen auf alle weiteren Inhalte einzuwirken. Zwar könnte dies auch über den `<body>`-Container selbst geschehen, dies jedoch nur mit Einschränkungen und Nebenwirkungen (beispielsweise, wenn Sie eine Gesamtbreite für die Inhalte vergeben wollen). Das Einschieben eines solchen **Zwischencontainers** ist daher gängige Praxis und sinnvoll. Sie können in seinem Inneren beliebig zusätzliche logische Bereiche durch weitere `<div>`-Hierarchien bilden. Dies bildet die Grundlage für ein Seitenlayout mit CSS, das deshalb auch als **Div-Layout** bezeichnet wird. Dazu später mehr. (Für alle im Folgenden vorgestellten Elemente gilt, dass sie auch *unmittelbar* in `<body>` stehen dürften.)

Überschrift <h1>

Angenommen, an dieser Stelle würde der Textinhalt des Dokuments beginnen (der umliegende logische Block dient nur zum Gruppieren), so ist hier die Verwendung einer **Überschrift Erster Ordnung** angebracht, die als **Gesamtüberschrift** für den Dokumentinhalt dient.

```
<h1>Aufbau des Dokumentrumpfes</h1>
```

Weitere Überschriften würden durch die niederrangigen Überschriftencontainer `<h2>`, `<h3>` etc. bis `<h6>` gebildet. Das Auslassen von Hierarchiestufen aus Präsentationsgründen (die Überschriften werden verschieden groß dargestellt) gilt als schlechter Stil.

Textabsatzcontainer <p>

Das »Arbeitspferd« der Inhaltsgestaltung ist der Textabsatzcontainer `<p>`, den Sie ja bereits kennengelernt haben. Er darf **Text und Inlinecontainer** enthalten, die ihrerseits wieder Text und Inlinecontainer enthalten können.

```
<p>Dieser Textabsatz enthält <i>Inlinecontainer</i> und Text. Das
<b>Prinzip</b> ist einfach. Im Block darf <span>Inlinecontent</span>
stehen. Blöcke können jedoch nicht immer verschachtelt werden.</p>
```

Der logische Blockcontainer `<div>` darf weitere Blockelemente enthalten, da er sonst seine Hauptaufgabe, das Gruppieren von Elementen, nicht erfüllen könnte. Dies gilt nicht für den Textabsatzcontainer, in dem **keine Blockelemente** erscheinen dürfen (sie würden den Absatz unterbrechen). Textabsätze können daher auch nicht verschachtelt werden.

Blockzitate <blockquote>

Ein Blockzitat `<blockquote>` kennzeichnet einen längeren, zitierten Textabschnitt. Es kann über ein Attribut `cite`, das hier nicht vorgestellt wird, auch auf die Quelle des Zitats verweisen.

```
<blockquote>Sein oder Nichtsein,
 das ist hier die Frage!</blockquote>
```

Blockzitate dürfen verschachtelt werden und können auch andere Blockelemente wie Textabsätze enthalten (nicht umgekehrt!). Da der Inhalt eines Zitatblocks eingerückt präsentiert wird, hat man ihn in früheren Zeiten speziell zum Erzielen von Einrückungen eingesetzt. Dies gilt jedoch als schlechter Stil.

Präformatierter Block <pre>

Der `<pre>`-Container ist eigentlich ein Relikt aus den Anfangstagen. In seinem Inneren ist die sonst durch den Browser vorgenommene Normalisierung von Leerzeichen und Zeilenumbrüchen deaktiviert (der Browser fasst sonst alle nebeneinanderliegenden entsprechenden Zeichen zu einem Leerzeichen zusammen).

Die Folge ist, dass der Inhalt des `<pre>`-Containers im Browser genauso mit allen Einrückungen, Tabulatoren und Umbrüchen präsentiert wird, wie er im Quelltext steht. Als Schrift wird Courier oder eine vergleichbare Schrift eingesetzt. Früher wurde er daher als Tabellenersatz verwendet.

```
<pre>   -----------------
        | 1 | 2 | 3 | 4 |
        ----------------- </pre>
```

Der `<pre>`-Container darf Inline-Elemente enthalten, allerdings nur solche, die nicht die Zeilenhöhe beeinflussen können. Bilder und Objekte sind daher verboten. Textformatierungselemente wiederum sind erlaubt.

Trennlinie <hr>

Auch die horizontale Trennlinie ist im Grunde ein Relikt, das man heutzutage sparsam oder gar nicht mehr einsetzen sollte. Der `<hr>`-Tag dient dazu, das Ende eines logischen Textabschnitts anzuzeigen, indem er beispielsweise zwischen zwei Textabsätze platziert wird. Seine Wirkung kann durch CSS-Anweisungen nachgeahmt werden.

```
<hr noshade="noshade" />
```

Als leeres Element wird der `<hr>`-Tag in XHTML entsprechend notiert. Auch sein leeres Attribut `noshade` muss mit einem Wert versehen werden.

Tabelle <table>

Auf Blockebene im `<body>` sind auch **Tabellen** erlaubt, was dazu verleitete, sie früher zum Layout von Dokumenten einzusetzen, indem beispielsweise Bereiche »oben« (Seitenkopf), »links« (Navigation) und »rechts« (Inhalt) durch Tabellenzellen erzeugt wurden (eine andere Möglichkeit, Inhalte nebeneinander zu platzieren, gab es anfangs nicht). Wegen seiner Unflexibilität und mangelhaften Barrierefreiheit ist das Tabellenlayout heutzutage verpönt. Zum Tabellieren von Daten, also der ursprünglichen Bestimmung des Elements, kann `<table>` allerdings eingesetzt werden.

Die hier gezeigte Tabelle besitzt ein `class`-Attribut, was sie als Teil einer Klasse namens »daten« kennzeichnet. Es dürfen mehrere Tabellen dieser Bezeichnung pro Dokument auftreten (was sinnvoll ist, da man ja vielleicht mehrere »Datentabellen« benötigt). Der Name »daten« ist willkürlich gewählt. Er kann dazu dienen, diese Gruppe von Tabellen mit CSS-Anweisungen speziell darstellen zu lassen.

```
<table class="daten">
   <tr><td>1</td><td>2</td></tr>
</table>
```

Im Inneren des `<table>`-Containers befinden sich **Tabellenzeilencontainer** `<tr>`, die ihrerseits **Datenzellen** `<td>` enthalten. Der Aufbau von Tabellen ist jedoch meist wesentlich komplexer als hier, weshalb ich hier auf die ausführliche Referenz in Abschnitt 14.79, *Elementreferenz,* verweisen möchte (zu den Unterelementen von `<table>` existieren ebenfalls ausführliche Erläuterungen).

Formular <form>

Ebenso wie Tabellen dienen Formulare als funktionale Bestandteile eines HTML-Dokuments. Der `<form>`-Container darf (mit Ausnahme von `<p>`) auch in anderen Blockelementen auftreten. Umgekehrt dürfen weitere Blockelemente innerhalb des `<form>`-Containers für die nötige Struktur sorgen. Man wird also selten so wie hier direkt die Formulareingabe-Elemente `<input>` unmittelbar im `<form>`-Container vorfinden.

```
<form action="/beispiel/cgi.php" method="get">
  <input type="text" name="eingabe"/> <input type="submit"/>
</form>
```

Die Hauptaufgabe des `<form>`-Containers, abgesehen davon, dass er die Eingabefelder enthält und so als »zu diesem Formular gehörig« kennzeichnet, ist, das Ziel des Datenversands der eingegebenen Daten (`action`) und die Art des Versands (`method`) zu bestimmen. Auch Formulare sind in der Regel komplexer als hier

gezeigt – so existieren etwa ein Dutzend verschiedene Typen von Eingabefeldern und weitere Hilfselemente zur Barrierefreiheit (dazu später selbstverständlich mehr).

Liste

Unmittelbar in `<body>` erscheinen dürfen Listencontainer beliebiger Art, von denen hier exemplarisch die `<ul>`-Liste vorgestellt wird (es gibt zwei weitere Listentypen im »strict«-Vokabular, `<ol>` und `<dl>`, sowie noch zwei weitere, `<dir>` und `<menu>` im »transitional«-Vokabular). Die wichtigste Verwendung von `<ul>`-Listen besteht heutzutage nicht mehr in der reinen Auflistung von Aufzählungspunkten, sondern in der Definition von Navigationsmenüs. Hierzu ist, um die gewünschte Präsentation zu erreichen, allerdings erheblicher Einsatz von CSS erforderlich, denn von Haus aus sehen Listen eher langweilig aus.

```
<ul id="liste">
   <li>Rosen</li>
   <li>Tulpen</li>
</ul>
```

Im Inneren der Liste definieren `<li>`-Container die einzelnen Listenpunkte. Will man eine weitere Liste als Unterliste verschachteln, so muss dies im Inneren eines `<li>`-Containers geschehen (`<ul>` darf direkt nur `<li>` enthalten, nicht unmittelbar weitere `<ul>`-Container). Die Definitionsliste `<dl>` enthält keine `<li>`-Elemente, sondern `<dt>`- und `<dd>`-Container für den erklärten Begriff und dessen Erläuterung

Änderungsmarkierung <del> und <ins>

Vorgestellt wird hier eine Marke für entfernte Inhalte. Diese werden jedoch nicht versteckt, sondern durchgestrichen präsentiert (es sollten damit ursprünglich Streichungen, beispielsweise in Verträgen, markiert werden). Ebenso wie die Einfügemarke `<ins>` ist `<del>` ein Zwitter, der sowohl (wie hier gezeigt) auf Blockebene erscheinen kann als auch innerhalb von Blockelementen. In einem `<p>`-Container eingesetzt verhält das Element sich wie ein Inlinecontainer.

```
<del><p>Das ist gestrichener Text.</p></del>
```

Als Inhalt sind beliebige Elemente (selbstverständlich auch Text) erlaubt.

Weitere erlaubte Inhalte von <body>

Nicht eigens vorgestellt werden muss hier der `<script>`-Container, der schon in Zusammenhang mit dem Dokumentkopf erläutert wurde. Im `<body>` darf allerdings auch ein `<noscript>`-Container auftreten, dessen Inhalt dann gezeigt wird,

wenn keine Scripte ausgeführt werden können, also der Inhalt des <script>-Containers nicht ausgewertet werden kann. Im Dokumentkopf hat <noscript> wiederum nichts verloren.

Weitere Erläuterungen zu diesem Element führen an dieser Stelle zu weit. Da das <noscript>-Verhalten aber hochinteressant ist, möchte ich auf die Erörterungen in Abschnitt 14.59, *object*, verweisen.

Übersicht über alle erlaubten Inhalte von <body>

Die folgende Tabelle enthält Kurzbeschreibungen aller Blockelemente, die im Rahmen der »strict«-DTD im <body>-Container erscheinen dürfen:[1]

Element	Rolle	Typ
...	bezeichnet einen logischen Block, Abschnitt oder Bereich	Blockcontainer
... .	Überschriften in von <h1> bis <h6> fallender Hierarchie; gewöhnlich fett und mit Abstand zum folgenden Abschnitt dargestellt; Schriftgrößen spiegeln von groß nach klein die Hierarchiestufen 1 bis 6 wider	Blockcontainer
...	bezeichnet einen Textabsatz	Blockcontainer
<blockquote> ... </blockquote>	Auszeichnung als Zitatblock; gewöhnlich eingerückt präsentiert	Blockcontaine

<pre>
...
</pre>
	bezeichnet Bereich, für den Leerzeichen, Umbrüche und Tabulatoren des Quelltextes für die Präsentation übernommen werden sollen	Blockcontainer

<hr> (in HTML) <hr /> (in XHTML)	»horizontal rule«; fügt eine horizontale Linie als Trennung zwischen zwei Abschnitten ein	Leeres Element; Blockcharakter
<table> ... </table>	definiert eine Tabelle	Blockcontainer
<form> ... </form>	Container für Formularelemente; bestimmt über Attribute das Formularverhalten	Blockcontainer
 ... 	»unordered list«; Container für eine ungeordnete Liste; Listenelemente werden durch Bullets gekennzeichnet	Blockcontainer

Tabelle 3.6 Übersicht über die im Dokumentrumpf unmittelbar erlaubten Elemente

1 Ich enthalte Ihnen nichts Wesentliches vor – die wenigen weiteren, in »transitional« potentiell erlaubten Elemente sind in der Praxis irrelevant.

Element	Rolle	Typ
`<ol> ... </ol>`	»ordered list«; Container für eine geordnete Liste; Listenelemente `<li>` erhalten Numerierung mit Ziffern oder Buchstaben	Blockcontainer
`<script> ... </script>`	Markiert einen Scriptcontainer	Inlinecontainer
`<noscript> ... </noscript>`	Enthält alternativen Inhalt, falls Scripte nicht interpretiert werden	Blockcontainer
`<ins> ... </ins>`	»inserted«; markiert einen Abschnitt als hinzugefügt	Inlinecontainer (hier: Block)
`<del> ... </del>`	»deleted«; markiert einen Abschnitt als gestrichen	Inlinecontainer (hier: Block)

Tabelle 3.6 Übersicht über die im Dokumentrumpf unmittelbar erlaubten Elemente (Forts.)

3.5 Die Doctype-Deklaration und ihre Bedeutung

Einem HTML-Dokument kann eine so genannte **Doctype-Deklaration** vorangestellt werden, die in diesem Fall vor dem Starttag des Wurzelelements `<html>` stehen muss. Die Deklaration ist in HTML optional, in XHTML obligatorisch – ihr Zweck ist es in beiden Fällen, unmissverständlich den Dokumenttyp, genauer gesagt, die **Dokumenttyp-Definition** (DTD) zu nennen, nach deren Grammatik das Dokument aufgebaut ist.

Was ist die Dokumenttyp-Definition?

Die DTD enthält Angaben über die im Dokument erlaubten Elemente, ihre Inhaltsmodelle und ihre Attribute und deren Werte. Erst anhand der DTD-Angabe ist eine **Validierung** (Typgültigkeitsüberprüfung) eines Dokuments möglich. Ein HTML-Dokument ohne Doctype-Deklaration ist daher nicht validierbar, ein XHTML-Dokument ist stets validierbar.

Historisch stammt die Doctype-Deklaration aus **SGML**; es handelt sich hier also nicht um HTML-Syntax. Bei der Deklaration handelt es sich daher auch nicht um einen Tag bzw. ein Element. Konsequenterweise ist die Doctype-Deklaration ebenso wenig Teil des Document Object Models (DOM) wie eine dieser potentiell noch vorangehende XML-Deklaration – beide stehen außerhalb des eigentlichen Dokuments.

Der Aufbau der Doctype-Deklaration

Prinzipiell sieht eine Doctype-Deklaration wie folgt aus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
 "http://www.w3.org/TR/REC-html40/strict.dtd">
```

Die Deklaration beginnt mit der Zeichenfolge `<!`, die jede SGML-Vereinbarung einleitet. Anschließend folgt der reservierte SGML-Bezeichner `DOCTYPE` (dieser muss, um erkannt zu werden, in Großbuchstaben geschrieben sein). Anschließend folgt der eigentliche Bezeichner der DTD:

```
<!DOCTYPE HTML ...
```

Dieser stellt gleichzeitig den **Namen des Dokumenttyps** (*document class*) wie auch den des **Wurzelelements** des Dokuments dar. Hier steht also für sämtliche Versionen von HTML und XHTML der Name `HTML`, wobei in Deklarationen für XHTML-Dokumente abweichend `html` (in Kleinbuchstaben) geschrieben werden muss, um die dort geltende Tagschreibweise widerzuspiegeln.

Auf den Typ-Bezeichner folgt die Angabe der **Art der Deklaration**:

```
<!DOCTYPE HTML PUBLIC ...
```

Man unterscheidet prinzipiell, ob die vorliegende Dokumenttypangabe auf einem öffentlichen Standard beruht, wie hier mit Keyword `PUBLIC`, oder ob direkt per URL auf eine in Dateiform einzulesende DTD verwiesen wird – in diesem Fall würde das Keyword `SYSTEM` lauten:

```
<!DOCTYPE HTML SYSTEM ...
```

Dieser zweite Fall ist in DTD-Deklarationen für HTML-Dokumente ungebräuchlich; hier wird man ausschließlich mit `PUBLIC`-Deklarationen zu tun haben. Diese sind formal wie folgt aufgebaut:

```
<!DOCTYPE HTML PUBLIC
  "public-identifier"  "system-identifier">
```

Eine vollständige `PUBLIC`-Deklaration setzt sich wie im oben gegebenen Beispiel aus zwei Angaben zusammen:

- Erstens eine formale Benennung der DTD (im Grunde ein URI), der so genannte **Public-Identifier**:

  ```
  "-//W3C//DTD HTML 4.0//EN"
  ```

- Zweitens einen URL, der auf eine DTD-Ressource in Dateiform zeigt – dieser wird als **System-Identifier** bezeichnet:

  ```
  "http://www.w3.org/TR/REC-html40/strict.dtd"
  ```

Der Public-Identifier allein soll einer Anwendung (in unserem Fall dem Browser) prinzipiell bereits genügen können, um die DTD zu identifizieren. Der System-Identifier-String ist daher in HTML im Prinzip optional. Er bezeichnet eine reale Ressource, die unter der angegebenen Adresse verfügbar ist (und gegebenenfalls von der Anwendung von dort bezogen werden kann).

Wie funktioniert der Public Identifier der Doctype-Deklaration?

Theoretisch könnte eine Anwendung über ein lokales Repository verfügen, in der DTDs gelagert sind, so dass der URI zu einem URL zur realen Ressource aufgelöst werden könnte. Ist der Public-Identifier nicht auflösbar, soll auf den System-Identifier zurückgegriffen werden. Da die HTML-DTDs in Browsern als implementiert angesehen werden können, ist ein System-Identifier zwar verzichtbar, seine An- oder Abwesenheit hat jedoch, wie in Folge gezeigt wird, dennoch Auswirkungen auf das Verhalten des Browsers.

Wo in XML zur Angabe eines Public-Identifiers das *gleichzeitige* Vorhandensein eines System-Identifiers zwingend ist, wird dies in HTML nicht so strikt gehandhabt – hier besteht die Möglichkeit, eine Doctype-Deklaration lediglich mit dem Public-Identifier vorzunehmen. Im HTML-Kontext kann man daher auch verkürzte PUBLIC-Deklarationen in folgender Form antreffen:

```
<!DOCTYPE HTML PUBLIC "public-identifier">
```

also konkret beispielsweise

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

Aufbau des Public-Identifiers

Die Zeichenkette eines formalen Public-Identifiers zerfällt in mehrere, durch Doppel-Slash getrennte Teilstrings:

1. Die Registriernummer der anmeldenden Organisation gemäß ISO 9070. Im Falle einer entsprechenden Registrierung steht hier der numerische »ISO Owner Identifier«, ansonsten ein Minuszeichen.
2. Der Name der anmeldenden Organisation
3. Angabe der Art der Ressource als DTD, TEXT, ENTITIES etc. (*public text class*) und deren Bezeichner (*public text description*).
4. Die Sprache (*public text language*), in der die Ressource abgefasst ist, in Form einer zweibuchstabigen Abkürzung gemäß ISO Standard 639.

Der Public-Identifier "-//W3C//DTD HTML 4.0//EN" bedeutet also im Detail:

- Er stammt von einer Organisation, die nicht gemäß ISO 9070 registiert ist.
- Der Name der Organisation ist »W3C« (bei älteren Deklarationen mag hier auch gelegentlich »IETF«, Internet Engineering Task Force, stehen).
- Es handelt sich um eine DTD, deren Name »HTML 4.0« ist.
- Die DTD ist in englischer Sprache abgefasst.

3.6 Beispiele für HTML-Grundstrukturen mit Doctype

Abschließend erfolgt eine kommentierte Übersicht über die derzeit gültigen Dokumenttypen und die sich daraus ergebende Grundstrukturen von Dokumenten. Für XHTML 1.1 wird auch auf die Möglichkeit der Einbindung einer XML-Schema-Deklaration eingegangen.

Anmerkung: Für die älteren Versionen HTML 2.0 und HTML 3.2 existieren ebenfalls Dokumenttyp-Deklarationen. Auf ihre Vorstellung wird an dieser Stelle verzichtet, weil diese Dokumenttypen zu Recht als veraltet gelten.

3.6.1 HTML-Dokumenttypen

HTML 4.01 transitional

Der liberalste HTML-Typ. Beachten Sie, dass im PUBLIC-Identifier das Schlüsselwort Transitional erscheint, die DTD selbst aber unter dem Dateinamen »loose.dtd« läuft (daher wird der Dokumenttyp ebenfalls manchmal als »loose« HTML bezeichnet.

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

Listing 3.1 Dokument HTML 4.01 transitional

HTML 4.01 strict

In der »strict«-Version taucht im `PUBLIC`-Identifier kein Schlüsselwort auf, da diese Version als die »reguläre« erachtet wird, von dem die anderen beiden die Abwandlungen darstellen. Die DTD selbst trägt den Dateinamen »strict.dtd«.

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Grundstruktur HTML 4.01 Strict</title>
  </head>
  <body>
    <!-- Dokumentinhalt -->
  </body>
</html>
```

Listing 3.2 Dokument HTML 4.01 strict

HTML 4.01 frameset

Die Frameset-Version zeigt, neben der anders lautenden Doctype-Deklaration anstelle des `<body>`-Elements ein `<frameset>`-Element, das in dieser Position als das »äußere« `<frameset>`-Element bezeichnet wird (in seinem Inneren dürfen neben `<frame>`-Tags auch weitere, »innere« `<frameset>`-Elemente erscheinen).

Als letztes Element *darf* im äußeren Frameset ein `<noframes>`-Container auftreten, der dann einen `<body>`-Container enthalten *muss*, der zum Frameset alternativen Inhalt zur Verfügung stellt.

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
  <head>
    <title>Grundstruktur HTML 4.01 Frameset</title>
  </head>
  <frameset>
  <!-- Frames -->
  <noframes>
    <body>
      <!-- Zum Frameset alternativer Inhalt -->
    </body>
  </noframes>
  </frameset>
</html>
```

Listing 3.3 Dokument HTML 4.01 frameset

3.6.2 XHTML-Dokumenttypen

Es existieren vier Versionen von XHTML 1.0 und 1.1, die alle den gleichen Namensraum besitzen (erst XHTML 2.0 wird durch einen anders lautenden Namensraum gekennzeichnet werden).

XHTML 1.0 transitional

Das XHTML 1.0 »transitional« Dokument entspricht dem Pendant in HTML 4.01. Die DTD-Datei heißt allerdings »xhtml1-transitional.dtd«.

```
<!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>Grundstruktur XHTML 1.0 Transitional</title>
  </head>
  <body>
    <!-- Dokumentinhalt -->
  </body>
</html>
```

Listing 3.4 Dokument XHTML 1.0 strict

XHTML 1.0 strict

Das Dokument von Typ XHTML 1.0 »strict« führt, im Gegensatz zu seinem Pendant in HTML 4.01 im `PUBLIC`-Identifier das Schlüsselwort `Strict`. Die Datei hat den Namen »xhtml1-strict.dtd«.

```
<!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>Grundstruktur XHTML 1.0 Strict </title>
  </head>
  <body>
    <!-- Dokumentinhalt -->
  </body>
</html>
```

Listing 3.5 Dokument XHTML 1.0 strict

XHTML 1.0 frameset

Die Frameset-Version von XHTML 1.0 entspricht derjenigen von HTML 4.01, sieht man von der Schreibweise der leeren Elemente ab. Auch hier erscheint als letztes Element im äußeren Frameset optional ein `<noframes>`-Container, der ein alternatives `<body>`-Element enthält.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Grundstruktur XHTML 1.0 Frameset</title>
  </head>
  <frameset>
    <!-- Frames -->
    <noframes>
      <body>
        <!-- Zum Frameset alternativer Inhalt -->
      </body>
    </noframes>
  </frameset>
</html>
```

Listing 3.6 Dokument XHTML 1.0 frameset

XHTML 1.1

Beachten Sie, dass sich der **Namensraum** des XHTML 1.1-Dokuments nicht von dem eines XHTML 1.0-Dokuments unterscheidet! Hier ist der Dokumentprolog noch durch eine vorangestellte **XML-Deklaration** erweitert.

Diese ist für XML-Dokumente optional und kann wie hier dazu dienen, mit dem `encoding`-Attribut den Zeichensatz des Dokuments zu nennen. Zu diesem Zweck kann jedoch auch die Information des vom Server übermittelten HTTP-Headers oder ein `<meta>`-Element im Dokumentkopf eingesetzt werden. Da keine Varianten vorkommen, heißt die DTD »xhtml11.dtd«.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Grundstruktur XHTML 1.1 </title>
  </head>
  <body>
    <!-- Dokumentinhalt -->
```

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

Listing 3.7 XHTML 1.1 Dokument

XHTML 1.1 mit Schema-Locator

Anstatt nur mit einer DTD kann die Grammatik eines Dokumenttyps auch mit einem XML-Schema festgelegt werden. Der Bezug auf ein bestimmtes Schema wird vom Dokument aus mit einem so genannten **Schema-Locator** deklariert. Dies geschieht durch das Attribut `xsi:schemaLocation`, das einen Link zur Schemadefinition darstellt. Zwangsläufig muss mit `xmlns:xsi` auch der Namensraum des Schemas deklariert werden. Beide Attribute werden im Starttag des Wurzelelements `<html>` notiert, der demzufolge (siehe Listing) ziemlich vollgestopft wirkt. Der Rest des Dokuments ist unverändert. Auch die nun eigentlich redundante Doctype-Deklaration muss, da vorgeschrieben, stehen bleiben.

Mit **Schema-Locator** sieht der XHTML 1.1-Dokumenttyp folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de"
 xmlns:xsi =
 "http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation =
 "http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd">
  <head>
    <title>Grundstruktur XHTML 1.1 mit XML-Schema</title>
  </head>
  <body>
    <!-- Dokumentinhalt -->
  </body>
</html>
```

Listing 3.8 XHTML 1.1 Dokument mit Angabe eines XML-Schemas

3.7 Zusammenfassung

Diese Kernaussagen sollten Sie aus diesem Kapitel mitnehmen:

- Ein HTML- oder XHTML-Dokument besteht aus einem **Wurzelelement**, das einen **Dokumentkopf** und einen **Dokumentrumpf** enthält.
- Die vier Elemente `<html>`, `<head>`, `<title>` und `<body>`, die zum Erstellen der Grundstruktur eines Dokuments dienen, nennt man **Strukturelemente**.

- Dokumente gemäß der Frameset-DTD haben einen anderen Grundaufbau.
- Der **Dokumentkopf** enthält **Metainformation** über das Dokument oder **funktionale Bestandteile** wie Scripte oder CSS-Anweisungen beziehungsweise Verweise auf entsprechende, extern gelagerte Dateien.
- Der **Dokumentrumpf** enthält die **sichtbaren Bestandteile** eines Dokuments, die in Form von **Blockelementen** im Inneren des `<body>`-Containers erscheinen. Inline-Elemente oder Text dürfen erst im Inneren dieser Blöcke auftreten.
- Die einem HTML-Dokument optional und einem XHTML-Dokument obligatorisch vorangestellte **Doctype-Deklaration** verweist auf eine DTD, gemäß der ein Dokument **validiert** werden kann (aber nicht muss).
- Die **Art der Deklarationen** wird durch den `PUBLIC`-Identifier genannt, der eine abstrakte Beschreibung, aber auch einen konkreten Verweis auf eine **DTD-Datei** enthält.

Wie geht es weiter?

Das folgende Kapitel beschäftigt sich mit der Grundsyntax und allgemeinen Regeln des Einsatzes von CSS sowie dessen Einbindung in HTML- und XHTML-Dokumente. Des Weiteren wird das CSS Box Modell vorgestellt und exemplarisch angewendet.

In diesem Kapitel erfolgt, nach der Darlegung des theoretischen Fundaments eine Einführung in die Syntax von CSS anhand einiger Beispiele. Demonstriert wird die Formulierung von Darstellungsregeln und deren Bindung an Elemente über interne und externe CSS-Angaben.

4 CSS Grundsyntax

Aufgrund der andersartigen Syntax stellen Stylesheets für angehende HTML-Autoren eine kleine Hürde dar, die aber nicht schwer zu nehmen ist. Wie bereits erläutert wurde, übernehmen Stylesheets die Aufgabe, die Darstellung eines Dokuments zu definieren, wohingegen HTML (in Reinform) oder XHTML ausschließlich für die Informationsstruktur zuständig sind.

4.1 Die Grundsyntax von CSS

Auch CSS ist **keine Programmiersprache**. Hier wird lediglich beschreibend festgelegt, welche Elemente auf welche Weise dargestellt werden und in welchem Kontext welche der definierten Darstellungsanweisungen einzusetzen ist. Man bezeichnet CSS daher als **Präsentationssprache**. Da kein Prozess, sondern ein Ergebnis beschrieben wird, nennt man es auch »deklarative« Sprache.

4.1.1 Verknüpfung von CSS-Anweisungen mit einem HTML-Dokument

Im vorigen Kapitel, in dem der Aufbau des Dokumentkopfes einer HTML-Seite erläutert wurde, fanden Sie bereits zwei Möglichkeiten, wie CSS-Anweisungen und ein HTML-Dokument zusammengebracht werden können: mit dem `<link>`-Element und mit dem `<style>`-Container.

Insgesamt existieren jedoch vier Wege, wie eine Styleanweisung in ein Dokument und letztlich zu einem zu präsentierenden Element gelangen kann (exemplarisch gezeigt in Abbildung 4.1). Werden alle vier Methoden gleichzeitig auf ein Dokument angewendet, so wirkt die **Summe** ihrer CSS-Anweisungen auf das Dokument (symbolisiert durch den Pfeil). Die Methoden sind der Reihe nach:

- externes, verlinktes Stylesheet
- externes, importiertes Stylesheet
- Dokumentstyle in `<style>`-Container
- Inlinestyle direkt im `style`-Attribut eines Elements

Diese Reihenfolge spiegelt gleichzeitig eine Art von **Rangordnung** wider. Setzt beispielsweise ein externes, verlinktes Stylesheet die Schriftfarbe eines Elements auf Rot und eine Anweisung innerhalb des Dokumentstyles die Schriftfarbe für das gleiche Element auf blau, so »gewinnt« der Dokumentstyle. Ebenso ist der Dokumentstyle ranghöher als importierte Anweisungen, die wiederum höherrangig als extern verlinkte Anweisungen sind.

Die einzige Styleanweisung, die nicht »ausgestochen« werden kann, ist der Inlinestyle. Dieser ist wegen seiner Nähe zum gestylten Element stets der ranghöchste. Sein prinzipieller Nachteil besteht darin, dass er auf gleicher Ebene angewendet wird, auf der sich der Dokumentinhalt befindet: Hier tritt wieder eine vermeidbare Vermischung von Struktur und Präsentation auf.

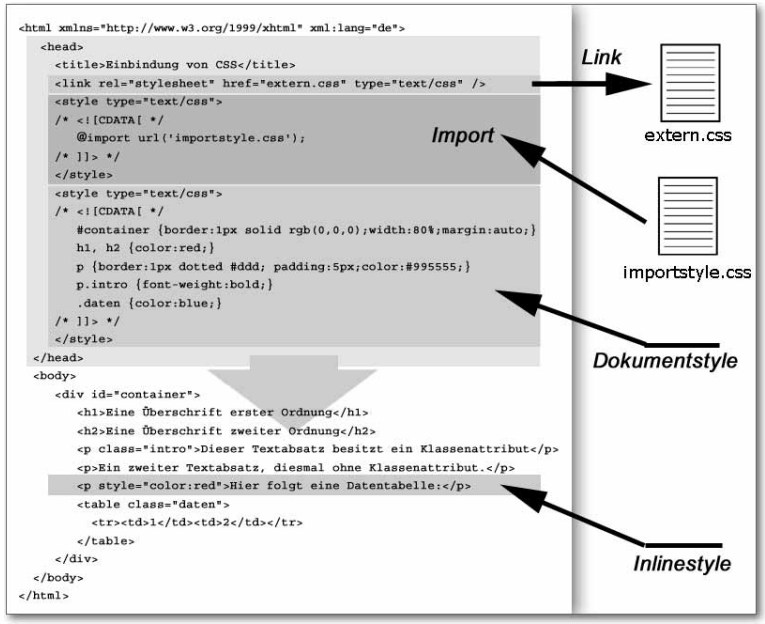

Abbildung 4.1 Mögliche Einbindung von CSS-Regeln in ein Dokument

Bevor ich auf die eigentliche Syntax der CSS-Anweisungen selbst eingehe, möchte ich zunächst die Einbindung der Anweisungen in das HTML-Dokument genauer ansprechen.

Verlinktes Stylesheet

Ein externes Stylesheet ist wie ein HTML-Dokument eine Textdatei, wenn auch mit anderer interner Syntax. Es besteht aus einer Abfolge von Styleanweisungen. Um diese auf eine Webseite anzuwenden, muss eine **Verknüpfung** zwischen beiden Dokumenten definiert werden. Dies geschieht mit dem `<link>`-Element. (Ich werde nachfolgend die XHTML-Syntax verwenden, aber wo erforderlich die Unterschiede zu HTML anmerken.)

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

Da das `<link>`-Element verschiedenen Zwecken dient, muss definiert werden, welche **Relation** zwischen dem Dokument und der externen Ressource besteht, auf die verwiesen wird. Dies geschieht mit dem `rel`-Attribut, das in diesem Fall den Wert `"stylesheet"` erhält.

Der **Verweis** selbst (eine Hyperreferenz) wird durch das `href`-Attribut ausgedrückt. Diesem werden als Wert der Dateiname und der Pfad zur Ressource übergeben. Hier befindet sich das Stylesheet im gleichen Verzeichnis wie die HTML-Seite, mit der es verknüpft wird.

Schließlich muss mit dem `type`-Attribut noch der **MIME-Typ** der Ressource genannt werden, der für CSS-Dateien stets `"text/css"` lautet. Alle Anweisungen des externen Stylesheets werden nun auf das Dokument angewendet.

Eine Hyperreferenz – wie funktioniert das mit dem Pfad?

Hyperreferenzen, also Verknüpfungen der aktuellen Seite mit einem anderen Dokument (aka »Links«), werden stets mit dem `href`-Attribut ausgedrückt. Die Verknüpfung entspricht einer Wegbeschreibung (Pfad) zum anderen Dokument, das durch seinen **Dateinamen** bezeichnet wird. Der **Pfad** kann ausgehend vom aktuellen Dokument beschrieben werden (relativer Pfad) oder von einem absoluten Bezugspunkt (absoluter Pfad).

Der **absolute Pfad** entspricht einer vollständigen Webadresse:

```
http://www.beispieldomain.de/verzeichnis/zieldatei.html
```

Der **relative Pfad** rechnet die aktuelle Position des Dokuments ein (den so genannten Basis-URI). Befindet sich das Dokument im Wurzelverzeichnis `http://www.beispieldomain.de/` und wird das gleiche Zieldokument wie eben angesprochen, so genügt die relative Angabe:

```
verzeichnis/zieldatei.html
```

Befindet sich das Ausgangsdokument parallel zum Zieldokument in einem anderen Verzeichnis `http://www.beispieldomain.de/woanders/`, so muss der Pfad zunächst in das **übergeordnete Verzeichnis** (mit `..`) wechseln und anschließend wieder **nach unten** in das Zielverzeichnis. Dieser relative Pfad sähe folgendermaßen aus:

Eine Hyperreferenz – wie funktioniert das mit dem Pfad?

```
../verzeichnis/zieldatei.html
```

In diesem Beispielen ist ein HTML-Dokument als Zieldatei verwendet. Das Prinzip für eine externe CSS-Datei ist jedoch identisch.

Importiertes Stylesheet

Der Stylecontainer `<style>` kann sowohl direkt Styleanweisungen beinhalten (dazu gleich) als auch eine oder mehrere **Importanweisungen**. Hierzu wird der Anweisung `@import` der URI des Zieldokuments übergeben. Das Prinzip der Pfad- und Dateiangabe entspricht dem bei einer verlinkten Datei:

```
<style type="text/css">
<![CDATA[
   @import url('importstyle.css');
]]>
</style>
```

In diesem Fall handelt es sich jedoch nicht um eine Verknüpfung, sondern, wie der Name schon sagt, um den **Import** von Anweisungen. Dies bedeutet, dass die extern liegenden Anweisungen gelesen und so behandelt werden, als ob sie physisch an Stelle der Importanweisungen stehen würden. Dies ist wichtig, denn falls mehrere Importanweisungen nacheinander stehen, so gilt im Konfliktfall die *zuletzt* erfolgte als maßgeblich.

Dokumentstyle

Enthält der Stylecontainer unmittelbar CSS-Anweisungen, so spricht man von einem **Dokumentstyle**, da sich diese Anweisungen ausschließlich auf das aktuelle Dokument beziehen. Der Stylecontainer benötigt ein `type`-Attribut für den **MIME-Typ** der Anweisungen, um die Interpretation der Daten durch den Browser zu steuern. Dem Reglement entsprechend muss der Containerinhalt in XHTML als `CDATA`-Bereich ausgezeichnet werden. In diesem hier ist exemplarisch eine Styleanweisung enthalten, die sich auf die Überschriften `<h1>` richtet:

```
<style type="text/css">
<![CDATA[

   h1  { color:red; }

]]>
</style>
```

Listing 4.1 Stylecontainer gemäß der XHTML-Spezifikation

Theorie und Praxis der Einbindung mit <style>-Container

In der Praxis verursacht die Einbindung mit einem CDATA-Bereich Schwierigkeiten – die enthaltenen Anweisungen werden nur teilweise oder gar nicht ausgewertet (abhängig vom Browser), weil die CDATA-Marken nicht der CSS-Syntax entsprechen, in die der Browser im Inneren des Containers umschaltet. Es empfiehlt sich daher, die Marken vor dem Browser zu verstecken. Dies erreicht man, indem man sie mit der Zeichenfolge /* ... */ umgibt:[1]

```
<style type="text/css">
 /* <![CDATA[ */

   h1  { color:red; }

 /* ]]> */
</style>
```

Listing 4.2 Stylecontainer in XHTML im praktischen Einsatz

Im Grunde sind die CDATA-Marken nur dann wirklich erforderlich, wenn das Dokument als XML ausgewertet wird, da die theoretische Möglichkeit besteht, dass CSS-Anweisungen im Inneren des Containers verbotene Zeichen oder Zeichenfolgen enthalten, die die Wohlgeformtheit des Dokuments zerstören. Dies liegt daran, dass das Innere von <style> als PCDATA-Text definiert ist, also geparst wird. Der CDATA-Bereich verhindert dies, indem der Inhalt als »ungeparste Zeichendaten« deklariert wird, der keinen Schaden anrichtet.

In HTML ist die Einbindung eines Dokumentstyles ähnlich einfach, wobei man hier nicht CDATA-Marken verstecken muss (diese sind in HTML nicht notwendig, da das Containerinnere von vornherein als CDATA definiert ist). Stattdessen werden die gesamten Styleanweisungen maskiert. Dies hat nicht den Hintergrund etwaiger Kompatibilitätsprobleme, sondern soll lediglich ältere Browser (die aber kaum noch anzutreffen sind) daran hindern, die CSS-Anweisungen im Browserfenster darzustellen. Hierfür genügt ein **HTML-Kommentar**:

```
<style type="text/css">
<!--

   h1  { color:red; }

-->
</style>
```

Listing 4.3 Stylecontainer in HTML im praktischen Einsatz

1 Dies entspricht der Kommentarsyntax von CSS, die nachfolgend erläutert wird.

Können Dokumentstyles und Importanweisungen gleichzeitig auftreten?

Will man sowohl Dokumentstyles als auch Importanweisungen einsetzen, so kann man dies durchaus auch im gleichen Stylecontainer tun. Es gilt:

- Alle Importanweisungen stehen zu Beginn.
- Dokumentstyleanweisungen folgen der letzten Importanweisung.

Der Grund für diese Reihenfolge liegt im »physischen Einfügen« der importierten Regeln. Dadurch, dass die Importanweisung vor den Dokumentstyles steht, stehen auch die importierten Regeln vor diesen, weshalb Dokumentstyles importierte Styles im Konfliktfall überschreiben. Man kann also **allgemeingültige** Regelsätze importieren und diese durch **dokumentspezifische** Regelsätze im Dokumentstyle überschreiben.

Inlinestyle

Allgemein missbilligt, wenn auch manchmal durchaus praktisch, ist die Anwendung so genannter Inlinestyles. Die Bezeichnung rührt daher, dass sie direkt im Starttag des zu formatierenden Elements, also »inline« definiert werden. Sie werden als Wert des `style`-Attributs übergeben. Dieses Attribut ist eines der Universalattribute, kann also für (beinahe) jedes Element eingesetzt werden.

```
<p style="color:red">Hier kommt eine Datentabelle:</p>
```

Ihr **Vorteil** besteht darin, dass man schnell, quasi »ambulant« die Darstellung einzelner Elemente festlegen kann, ohne sich Gedanken darüber zu machen, ob bereits andere CSS-Anweisungen dafür existieren (der Inlinestyle ist stets höherrangig). Der **Nachteil**, sofern man Inlinestyles nicht wirklich nur punktuell einsetzt, besteht darin, dass das Dokument schwerer zu warten ist – bei Änderungen muss stets der Dokumentrumpf nach eventuell vorhandenen Inlinestyles durchsucht und diese geändert werden. Außerdem stehen Präsentationsanweisungen dann dort, wo sie nicht hingehören: im Dokument selbst.

4.1.2 Aufbau einer CSS-Anweisung

Ein Stylesheet ist nichts weiter als eine Abfolge von **Formatierungsregeln** (schematisch in Abbildung 4.2 zu sehen), so genannten »rule sets«. Jede Regel zerfällt in zwei Teile:

- Den **Selektor**
  Dieser weist die Präsentationsregel über ein Vergleichsmuster einem Elementcontainer eindeutig zu:

- Den **Deklarationsblock** (»declaration block«)
  Dieser enthält, in geschweifte Klammern eingeschlossen, die eigentlichen Formatierungsanweisungen (Eigenschaftsdeklarationen).

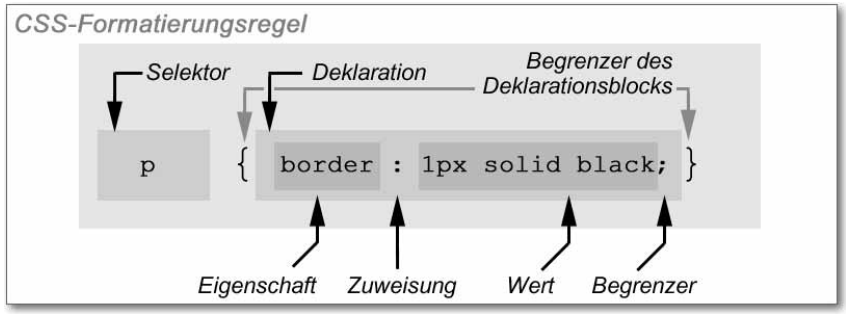

Abbildung 4.2 Schema einer CSS-Formatierungsregel

4.1.3 Die Deklaration einer CSS-Eigenschaft

Jede **Eigenschaftsdeklaration** ist zusammengesetzt aus Paaren von Eigenschaftsbezeichner (*name*) und Eigenschaftswert (*value*) – getrennt durch einen Doppelpunkt, der als Zuweisungsoperator dient. Leerzeichen um den Doppelpunkt sind erlaubt. Das Ende der Wertzuweisung wird durch ein abschließendes **Semikolon** gekennzeichnet. (Ein zugewiesener Wert kann daher selbst auch Leerzeichen enthalten oder die Form einer kommagetrennten Liste annehmen.)

Ein **Deklarationsblock** kann eine oder mehrere Wertzuweisungen (Einzeldeklarationen) enthalten, die zusammen mit dem Selektor die Formatierungsregel bilden. Der Deklarationsblock wird durch **geschweifte Klammern** begrenzt.

Leerzeichen und Zeilenumbrüche zwischen den Regeln sind irrelevant. Sie können Styleanweisungen daher beliebig durch Einrückungen und Zeilenumbrüche formatieren und so ihre Lesbarkeit erhöhen. Hier eine Regel, die drei Anweisungen enthält:

```
p  { border:1px dotted #ddd; padding:5px; color:#995555; }
```

Dieselbe Regel kann – etwas übersichtlicher – auch über mehrere Zeilen verteilt werden. Es ist Geschmackssache, ob Sie die schließende Klammer in eine neue Zeile setzen oder nicht:

```
p  { border:1px dotted #ddd;
     padding:5px;
     color:#995555;
}
```

Achtung – das schließende Semikolon nicht vergessen!

Ein vergessenes Semikolon nach einer Wertzuweisung führt dazu, dass diese und die auf sie folgenden Zuweisungen nicht beachtet werden (Syntaxverstoß). Die vorangegangenen Zuweisungen werden immerhin erkannt. Nur bei der letzten Zuweisung darf auf das beendende Semikolon verzichtet werden.

4.2 Selektoren und Style-Einbindung

Selektoren dienen dazu, eine CSS-Formatierungsregel an ein Element oder eine Gruppe von Elementen zu binden. Sie stehen daher im Zentrum der Funktionalität von Stylesheets. Die wichtigsten Grundformen werden in den folgenden Beispielen demonstriert.

Die Beispieldatei für dieses Kapitel: Sie finden auf der Begleit-CD im Ordner **Buchdaten/Kapitel_4** die Datei »beispiel1.html«, die unformatiertes HTML enthält (es handelt sich um ein XHTML *strict* Dokument), was ohne CSS etwas karg wirkt.

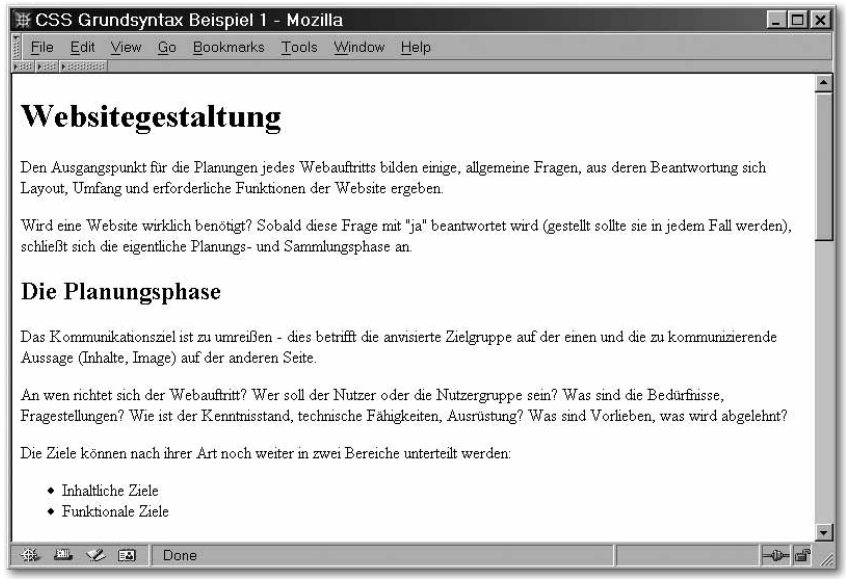

Abbildung 4.3 Die unformatierte Datei »beispiel1.html« im Browser

4.2.1 Der Selektor – Die Grundformen

Die folgenden Arbeitschritte erfolgen, sofern nichts anderes erwähnt wird, im derzeit noch leeren **Dokumentstyle-Container** des HTML-Dokuments. Fügen Sie

dort einfach die gleich beschriebenen Styleanweisungen ein. Etwas später wird jedoch auch ein externes, verlinktes Stylesheet ins Spiel kommen.

Der Typselektor

Im einfachsten Fall besteht ein Selektor aus einem Elementnamen. Der Typselektor zielt auf Elemente des genannten Tagnamens (*generic identifier*). Er besteht daher im einfachsten Fall aus einem Tagbezeichner. Eine solche Regel gilt global im gesamten Dokument für alle Elemente des bezeichneten Typs.

Die folgende Stylesheetregel betrifft alle `<h1>`-Container des Dokuments:

```
h1  { color:red; }
```

Die Anweisung besagt, dass die **Schriftfarbe** im Inneren des Containers rot sein soll. Der Eigenschaft `color`, die die Schriftfarbe (auch »Vordergrundfarbe« genannt) bezeichnet, wird hierfür ein Farbname zugewiesen. Statt des Farbnamens können (nein, sollten Sie besser) auch Hexadezimalangaben verwenden, die flexibler handhabbar sind. Farbwerte erhalten Sie aus Tabellen oder mit Hilfe eines Grafikprogramms (z. B. Photoshop) oder eines Color-Picker-Tools.

```
h1  {color:#ff0000; }
```

Diese Angabe hat exakt die gleiche Wirkung – die Überschrift `<h1>` wird rot.

Anmerkung: Eine Übersicht über Farbnamen und ihre Hexadezimalwerte finden Sie nach Farbtönen geordnet in Kapitel 18*, Farbwertereferenz*.

Gruppierung von Selektoren

Sie haben bemerkt, dass außer der Überschrift `<h1>` nichts beeinflusst wird – der Selektor »passt« auch auf keines der anderen Elemente. Soll die gleiche Anweisung auf mehrere Elemente wirken, kann man sie entweder mehrmals mit verschiedenen Selektoren wiederholen (nicht sehr ökonomisch) oder eine **Selektorgruppe** bilden.

Wie gruppiert man Selektoren?

Soll die gleiche Regel für mehrere Selektoren gelten, so können diese gruppiert werden. Hierfür werden die Einzelselektoren zu einer durch Komma getrennten Liste zusammengefügt. Im einfachsten Fall ist es eine **Liste aus Typselektoren**.

Die Überschriften `<h2>` sollen ebenfalls rot werden:

```
h1, h2  { color:#ff0000; }
```

In diesem Fall sind zwei Überschriften betroffen. Schreiben Sie *hinter* der ersten nun noch eine zweite Styleregel für die Überschriften `<h1>` bis `<h3>`. Diesmal soll die Schriftart (`font-family`) geändert werden. Es soll Schrift vom Typ »Arial« verwendet werden:

```
h1, h2, h3  { font-family:arial, helvetica, sans-serif; }
```

Hier werden sogar drei Schriften angegeben. Die zweite und dritte dienen als Ausweichschriften, falls Arial nicht zur Verfügung steht. Der allgemeine Typ `sans-serif` sorgt im äußersten Fall dafür, dass wenigstens eine serifenlose Schrift eingesetzt wird. Hier wird auch deutlich, dass das Semikolon wichtig ist, wenn wie hier im zugewiesenen Wert Leerzeichen enthalten sind.

Sie sehen nun zweierlei:

1. Es können mehrere Anweisungen auf das gleiche Element zielen.
2. Diese Anweisungen addieren sich.

In diesem Fall werden die Überschriften `<h1>` als auch `<h2>` sowohl rot als auch mit Arial dargestellt. Für `<h3>` gilt lediglich die Schriftart. Ergänzen Sie nun eine Farbangabe in der zweiten Anweisung:

```
h1, h2, h3  { font-family:arial, helvetica, sans-serif;
              color:#4994CD; }
```

Den drei Überschriften wird nun ein Blauton zugewiesen. Was geschieht? Die Überschriften `<h1>` und `<h2>` sind nun blau, obwohl für sie auch die Anweisung »rot« gilt (dass `<h3>` blau wird, ist weniger verwunderlich). Vertauschen Sie jetzt die beiden Styleregeln, so dass die Anweisung »rot« als zweite steht:

```
h1, h2, h3  { font-family:arial, helvetica, sans-serif;
              color:#4994CD; }
h1, h2      { color:#ff0000; }
```

Die Überschriften `<h1>` und `<h2>` sind nun wieder rot. Dies liegt daran, dass Styleanweisungen sich addieren, aber auch gegenseitig überschreiben können. In einem Konfliktfall »gewinnt« die ranghöhere Regel.

In diesem Fall besitzen beide Regeln einen Typselektor (es ist irrelevant, ob eine Gruppierung vorliegt – es wird jeweils der Konflikt vom zu stylenden Element aus betrachtet). Es gilt das Prinzip des »letzten Wortes«: Die als letzte stehende Regel überschriebt die vorangehende.

- Bei Konflikten überschreibt die höherrangige Anweisung die niederrangige.
- Bei gleichrangigen Anweisungen gewinnt die zuletzt stehende.

Sie können jetzt die »rot«-Anweisung entfernen, wenn Sie wollen. Den aktuellen Stand finden Sie in der Datei »beispiel1_2.html«.

Klassenselektor ohne Typbindung

Fügen Sie nun als Vorbereitung eine Anweisung für den `<p>`-Container hinzu, der dessen Schrift ebenfalls blau macht:

```
p  {color:#4994CD; }
```

Manchmal sollen Anweisungen nicht pauschal allen Elementen einer Sorte zugewiesen werden (wie eben geschehen), sondern nur einigen ausgewählten. Diese müssen, um als Ziel bestimmter Anweisungen kenntlich zu sein, jedoch gekennzeichnet werden. Weil sie als Untergruppe quasi als eine eigene Klasse bezeichnet werden können, erhalten sie ein Attribut `class`. Im Dokument finden Sie ein paar Absätze, die folgendermaßen beginnen:

```
<p class="intro">Den Ausgangspunkt f&uuml;r die Planungen jedes
Webauftritts bilden ... </p>
```

Um auf ein Element mit diesem Klassenattribut zuzugreifen, verwendet man eine CSS-Klasse. Diese wird dementsprechend mit einem Klassenselektor versehen, der aus dem Namen der Klasse besteht (achten Sie auf den Punkt vor dem Bezeichner).

```
.intro  { font-weight:bold;
}
```

Eine solche Klasse wirkt auf jedes Element mit entsprechendem `class`-Attributwert, ungeachtet des Elementtyps. Eine Klasse ohne Elementbindung bezeichnet man als **freie Klasse**. Sie können sie versuchsweise einem der `<li>`-Container weiter unten im Dokument zuweisen, indem Sie ihm ein Klassenattribut geben:

```
<ul>
<li class="intro">Inhaltliche Ziele</li>
<li>Funktionale Ziele</li>
</ul>
```

Sie sehen hier wieder zwei Dinge:

1. Eine freie Klasse wird jedem Element mit entsprechendem Klassenattribut zugewiesen, ungeachtet des Elementtyps.
2. Die speziellen Anweisungen der Klasse addieren sich zu den allgemeinen für den jeweiligen Elementtyp.

Sowohl der -Container als auch die <p>-Elemente mit Klassenattribut werden mit fetter Schrift dargestellt. Die Schrift der <p>-Container bleibt zusätzlich blau.

Klassenselektor mit Typbindung

Die Wirkung eines Klassenselektors kann jedoch auch auf eine Elementgattung beschränkt werden, indem ihm ein Typselektor vorangestellt wird. Der Punkt dient nun als Verbinder zwischen den beiden Teilen des Selektors.

```
p.frage  { font-style:italic;
}
```

Diese Anweisung gilt nur für <p>-Container mit dem Klassenattributwert "frage", wobei die Anweisung font-style:italic die Schrift kursiv macht. Man spricht von einer **typgebundenen Klasse**. Die entsprechenden <p>-Elemente werden, wie gewünscht, kursiv dargestellt. Versuchen Sie dies nun mit dem zweiten der -Container:

```
<ul>
<li class="intro">Inhaltliche Ziele</li>
<li class="frage">Funktionale Ziele</li>
</ul>
```

Dieser wird (wie Sie aber sicher erwartet haben) *nicht* gestylt. Erweitern Sie jetzt die Styledefinition um die Anweisung, die Schrift rot zu machen:

```
p.frage  { font-style:italic;
           color:#ff0000;
}
```

Die Schrift der <p>-Container wird in der Tat rot und kursiv. Im Listenelement geschieht nach wie vor nichts. Damit werden zwei Dinge deutlich:

1. Eine typgebundene Klasse wirkt nur auf die Elemente ihres Typs, auch wenn die Klasse (versehentlich) einem anderen Element zugewiesen wird.
2. Die speziellen Anweisungen einer Klasse überschreiben die allgemeinen Anweisungen für den durch die Klasse gestylten Container.

Der Klassenselektor ist »passgenauer« für ein Element mit einer Klasse als der Typselektor und wird daher als höherrangig eingestuft. Man sagt hierzu, der Klassenselektor habe die höhere **Spezifizität**.

Anmerkung: Den aktuellen Stand finden Sie in der Datei »beispiel1_3.html«.

Der ID-Selektor

Sie haben sicherlich bereits den `<div>`-Container bemerkt, der als Zwischencontainer in `<body>` den restlichen Seiteninhalt umfasst. Dieser besitzt ein `id`-Attribut, das ebenso wie ein Klassenattribut verwendet werden kann, um eine Styleanweisung an ein Element zu binden. Da ein `id`-Attribut nur einmal pro Dokument mit demselben Wert erscheinen darf, setzt man es ein, um funktionale (also einmalige) Bestandteile des Dokuments zu markieren.

In diesem Falle ist es ein »Inhaltsbereich«:

```
<body>
  <div id="inhalt">
    <h1>Websitegestaltung</h1>
    <p class="intro">Den Ausgangspunkt...
     ...
  </div>
</body>
```

Der ID-Selektor wirkt also immer auf *genau ein Element*, das ein `id`-Attribut mit dem übergebenen Bezeichner besitzt. Hierfür wird der ID-Wert mit vorangestelltem Hash-Zeichen eingesetzt:

```
#inhalt  { width:70%;
           border:1px dotted #4994CD;
           padding:1em;
}
```

Sie könnten hier, unter Hinzunahme des Elementbezeichners, auch genauer `div#inhalt` schreiben, was die Anweisungen unmissverständlicher macht (allerdings nur für menschliche Leser). Da es jedoch nur ein Element mit diesem ID geben kann, ist dies im Grunde überflüssig.

Da ein **Dokumentbereich** gestylt wird, soll er auch wie ein Bereich behandelt werden. Daher bekommt er zunächst Abmessungen, also eine Breite zugewiesen. Sie haben sicherlich bemerkt, dass der Text bislang die ganze Breite des Browserfensters einnahm. Das ist das normale Verhalten für Blockelemente wie `<h1>`, `<div>` oder `<p>`. Durch eine Breitenangabe mit `width` kann die Breite nun begrenzt werden.

Des Weiteren wird ein **Rahmen** mit `border` um das Element gelegt, der einen Rahmenstil `dotted` und eine Farbe erhält. Die Rahmenbreite ist auf 1 Pixel festgelegt. Damit der Rahmen nicht an den Inhalt stößt, wird mit `padding` ein Innenabstand festgelegt. Die Maßeinheit `em` bezieht sich auf die hier gültige Schriftgröße – hier entspricht `1em` etwa 16 Pixel.

Abbildung 4.4 Der Inhaltsbereich mit Rahmen und Padding (beispiel1_4.html)

[»] *Hinweis:* Beachten Sie den, in Abbildung 4.4 sichtbaren Abstand zwischen dem Containerrahmen und dem Rand des Browserfensters.

- ID-Selektoren erfassen genau einen Elementcontainer und werden für funktionale Dokumentbereiche oder -teile eingesetzt.
- Ein Bereich kann Abmessungen und weitere Eigenschaften erhalten.

4.2.2 Einbindung eines externen Stylesheets

Nun soll zusätzlich zum Dokumentstil ein externes Stylesheet hinzugenommen werden. Hierzu setzen Sie im Dokumentkopf (vor dem Style-Container) ein `<link>`-Element ein (dies geschieht in »beispiel1_5.html«):

```
<link rel="stylesheet" href="extern_bsp1.css" type="text/css"/>
```

Wenn Sie die CSS-Datei im Editor öffnen, sehen Sie, dass sie noch leer ist. Tragen Sie ein paar Anweisungen ein:

```
/* externes Stylesheet für Beispiel 1 */
body  { margin:0; padding:0; }
```

Dies setzt den Innenabstand dSes <body>-Elements auf Null, der in Abbildung 4.4 sichtbar in Erscheinung tritt.[2] Geben Sie jetzt zusätzlich einen Schrifttyp an:

```
/* externes Stylesheet für Beispiel 1 */
body  { margin:0; padding:0;
        font-family:arial, helvetica, sans-serif; }
```

Wie Sie sehen, wird nun der gesamte Dokumentinhalt in Arial dargestellt, auch für diejenigen Elemente, für die dies nicht eigens vereinbart wurde: Der <body>-Container reicht die ihm zugewiesenen Eigenschaften an die innen liegenden Container weiter. Dieses Prinzip wird als **Vererbung** bezeichnet.

Nun sollen weitere Eigenschaften definiert werden. Zum einen soll die Überschrift <h1> ein Hintergrundbild erhalten und näher an den oberen Rand des Inhaltscontainers gerückt werden. Auch der <body> erhält ein Hintergrundbild. Die folgenden Anweisungen kommen wieder in das externe Stylesheet:

```
h1 {background:url('h_bg.gif') repeat-y;
margin-top:0; padding:5px; }
```

Die background-Property weist über url() eine Hintergrundgrafik zu, die vertikal wiederholt wird (repeat-y). Zusätzlich bekommt die Überschrift einen Innenabstand, um den Containerbereich zu vergrößern. Die Anweisung margin-top:0 nullt den Abstand nach oben (das Padding des Inhalts gilt aber noch!).

Abbildung 4.5 Die Überschrift mit Hintergrundgrafik

Auch der <body> erhält eine Hintergrundgrafik. Da diese mit dem Textinhalt des Dokuments in Konflikt kommt, muss nun auch der Textbereich eine Hintergrundfarbe erhalten. Zudem soll er wieder auf einen kontrollierten Abstand zum Fensterrand gebracht werden. Die Anweisungen des externen Stylesheets sehen nun insgesamt folgendermaßen aus:

```
/* externes Stylesheet für Beispiel 1 */

body    { margin:0; padding:0;
          font-family:arial, helvetica, sans-serif;
          background:url('body_bg.gif') repeat-y;
}

h1      { background:url('h_bg.gif') repeat-y;
```

2 Nach den Spezifikationen würde margin:0 eigentlich ausreichen. Ältere Opera-Browser möchten gerne zusätzlich auch die Angabe padding:0.

```
          margin-top:0; padding:5px;
}

#inhalt { margin:1em auto 2em 12em;
          background-color:#fff;
}
```

Aufräumarbeiten

Es liegen nun interne Dokumentstyles und externe Angaben aus dem verlinkten Stylesheet vor, die zum Teil redundant sind. Es empfiehlt sich, dies zusammenzufassen, wofür sich das externe Stylesheet am besten eignet, da es im Zweifelsfall auch auf andere Dokumente angewendet werden kann, in denen die gleichen Anweisungen benötigt werden. Wollte man hierfür Dokumentstyles einsetzen, so müsste jeweils der Containerinhalt kopiert werden – es besteht die Gefahr von Inkonsistenzen, sobald Änderungen erforderlich sind.

Die Styles werden daher im externen Stylesheet zusammengeführt und der Dokumentstyle gelöscht. Die Farbgebung für den `<p>`-Container wird dabei entfernt (es genügt, wenn die Überschriften blau bleiben, für den Text ist der Farbkontrast zum Hintergrund zu schwach.) Nach der Zusammenführung sieht das externe Stylesheet folgendermaßen aus:

```
/* externes Stylesheet für Beispiel 1: */

body       { margin:0; padding:0;
             font-family:arial, helvetica, sans-serif;
             background:url('body_bg.gif') repeat-y; }

h1, h2, h3 { color:#4994CD; }

h1         { background:url('h_bg.gif') repeat-y;
             margin-top:0; padding:5px; }

#inhalt    { width:70%;
             border:1px dotted #4994CD;
             padding:1em;
             margin:1em auto 2em 12em;
             background-color:#fff; }

.intro     { font-weight:bold; }

p.frage    { font-style:italic;
             color:#ff0000; }
```

Listing 4.4 Die komplettierte CSS-Datei (extern_bsp1_5.css)

Abbildung 4.6 Der Stand der Dinge (beispiel1_5.html)

4.2.3 Welcher Selektor – wann, wofür?

Die **Wahl des Selektors** bestimmt sich aus Art, Verwendung und Häufigkeit des Auftretens des Zielelements. Wird ein Elementtyp pauschal formatiert? Oder nur unter bestimmten Umständen? Wie häufig tritt dieser Umstand auf?

So wird der **Typselektor** für pauschale Formatierungsanweisungen an eine bestimmte Elementgattung eingesetzt, ein **Klassenselektor** für eine Untergruppe von Elementen, die jedoch mehrfach auftreten kann. Der **ID-Selektor** schließlich wird, da es maximal ein Zielelement geben kann, zum Stylen funktionaler Bestandteile der Seite eingesetzt, die nur einmal vorhanden sind (»Inhaltsbereich«, »Navigationsbereich« etc.).

In der folgenden Tabelle finden Sie eine Übersicht.

Selektor	Verwendung	Beispiel
Typselektor	Formatierung aller Elemente eines Typs, pauschal	`p`
Typselektoren, gruppiert	Mehrere Elemente mit gleicher Formatierung; pauschal	`h1, h2, h3`
Klassenselektor, frei	Elemente beliebigen Typs mit entsprechendem Klassenattribut, wiederholbar	`.daten`

Tabelle 4.1 Grundtypen von Selektoren und ihre Einsatzgebiete

Selektor	Verwendung	Beispiel
Klassenselektoren, frei, gruppiert	Elemente beliebigen Typs mit einer der genannten Klassen, wiederholbar	`.a1, .a2, .a3`
Klassenselektor, gebunden	Formatierung von Elementen eines festgelegten Typs mit entsprechendem Klassenattribut, wiederholbar	`p.intro`
ID-Selektor	Formatierung genau eines Elements beliebigen Typs (Typnennung möglich, aber irrelevant) mit entsprechendem ID-Attributwert, nicht wiederholbar	`#container` `div#container`

Tabelle 4.1 Grundtypen von Selektoren und ihre Einsatzgebiete (Forts.)

Anmerkung: Es existieren weitere Selektortypen, die teilweise aus den bereits bekannten Typen zusammengesetzt sind (daher »zusammengesetzte Selektoren« genannt) oder die auf Zustände von Elementen oder Teile ihres Inhaltes abzielen. Diese Selektoren werden in den späteren Kapiteln erörtert.

4.2.4 Kommentare in Stylesheets

Auch in Stylesheets besteht die Notwendigkeit, Erläuterungstexte zu den Anweisungen hinzufügen zu können, ohne dass diese selbst als Anweisungen missverstanden werden oder einen Syntaxfehler verursachen. Einen solchen Kommentar leitet man durch eine bestimmte Zeichenfolge ein und beendet ihn mit einer weiteren wieder:

```
/* Dies ist ein Kommentar. Und hier ist er zu Ende: */
```

Die Zeichenfolge `/*` bewirkt, dass Browser nicht versucht, den nun folgenden Text als Styleanweisung zu interpretieren. Die Zeichenfolge `*/` wiederum beendet den **Kommentarmodus**. Mit Kommentaren lassen sich zweierlei Dinge anstellen. Zum einen kann man den Styleanweisungen Erläuterungen hinzufügen:

```
/* Diese Regel gilt für <p>-Container: */
p  { border:1px dotted #ddd;
     padding:5px;
     color:#995555;
}
```

Zum anderen lassen sich komplette Regeln oder Teile von ihnen auskommentieren, indem man sie ganz oder zum Teil in Kommentare einschließt:

```
/* Hier ist die padding-Angabe einzeln auskommentiert: */

p  { border:1px dotted #ddd;
```

```
 /*  padding:5px;   */
     color:#995555; }

/* Diese Regel ist komplett stillgelegt:
   h1, h3     { color:red; }
*/
```

Beides funktioniert. Dennoch: Das Auskommentieren von Regeln oder Teilen von Regeln ist eher als Notbehelf zu sehen und sollte nicht zur Gewohnheit werden. Spätestens wenn Kommentare versehentlich verschachtelt werden, ist Chaos vorprogrammiert:

```
/* Diese Regel soll nun doch komplett stillgelegt werden:

p  { border:1px dotted #ddd;
 /*  padding:5px;   */
     color:#995555; }
*/
```

Der Beginn des »inneren« Kommentars wird nicht beachtet (der Browser ist bereits im Kommentarmodus), seine Endmarke allerdings als Abschluss des äußeren Kommentars missverstanden. Der nun für den Browser sichtbare Rest verursacht einen Syntaxfehler.

4.2.5 Cascading, Addition und Konfliktregelung bei CSS-Anweisungen

Wie schon erwähnt, werden Styleanweisungen abhängig vom Ort ihrer Deklaration unterschiedlich gewichtet. Solange kein Konflikt zwischen Regeln auftritt, ist dies unerheblich, da die Regeln sich in diesem Falle grundsätzlich addieren. Tritt ein Konflikt auf, so überschreibt hingegen die höherrangige Regel die mit dem niedrigeren Rang.

4.2.6 Was bedeutet das »cascading« in Cascading Stylesheets?

Die **Addition von Styleregeln** erfolgt nach einem eigentlich simpel erscheinenden Prinzip. Es gelten grundsätzlich alle vorhandenen, auf ein Element irgendwie anwendbaren Formatierungsanweisungen aller Regeln. Hierbei müssen jedoch gelegentlich Konflikte zwischen konkurrierenden Anweisungen geregelt werden. Die Methode der Addition, Rangeinstufung und Konfliktlösung wird als **Kaskadierung** (cascading) bezeichnet.

Die Rangeinstufung richtet sich nach zwei Punkten:

- Nach der »Entfernung« der Deklaration der Regel vom Zielelement
  In diesem Sinne gilt die Regel als höherrangig, die »näher« an ihrem Zielelement definiert ist. Ein Inlinestyle ist bei Konflikten der ranghöchste.
- Nach ihrer »Passgenauigkeit«, der so genannten **Spezifizität**
  Eine Regel ist dann höherrangiger als eine andere, wenn ihr Selektor das Zielelement und den Kontext, in dem es sich befindet, besser beschreibt. Ein ID-Selektor ist spezifischer als ein Klassenselektor, dieser wiederum spezifischer als ein Typselektor.

Die Summe aller auf ein HTML-Dokument anzuwendenden CSS-Regeln nach ihrer Zusammenführung und der Regelung möglicher Konflikte wird als **Stylesheet** bezeichnet.

4.2.7 CSS-Eigenschaften

In CSS existieren zahlreiche Eigenschaften (Properties) denen Werte zugewiesen werden können. Man unterscheidet (u.a.) folgende Gruppen, aus denen in diesem Kapitel bereits einige Properties exemplarisch eingesetzt wurden:

Text-Properties

Diese Properties sind auf alle Elemente anwendbar, die direkt oder indirekt über Textinhalt verfügen. Sie regeln Eigenschaften, die nicht an den Schrifttyp gebunden sind, wie Abmessungen oder Abstände im Zeilenkontext. Hier eine Auswahl der wichtigsten Text-Properties:

- `letter-spacing` (Abstand zwischen den Zeichen eines Textes)
- `line-height` (Zeilenhöhe innerhalb eines Containers)
- `text-align` (horizontale Ausrichtung des Textes)
- `text-decoration` (Präsentation, ohne/unter-/durch-/überstrichen)
- `text-indent` (rückt die erste Zeile des Textinhalts ein)
- `text-transform` (festzulegende Schreibweise, klein/groß)
- `vertical-align` (vertikale Ausrichtung innerhalb der Zeile)
- `word-spacing` (Wortabstand zwischen einzelnen Worten)

Font-Properties

Die Font-Properties sind ebenfalls auf alle Elemente anwendbar, die direkt oder indirekt über Textinhalt verfügen, zielen in diesem Fall jedoch auf die Schriftfor-

matierung oder die Schriftart selbst ab (Font, Schnitt). Hier eine Auswahl der wichtigsten Font-Properties:

- `font` (alle Schriftpräsentations-Properties gemeinsam)
- `font-family` (Schriftart)
- `font-size` (Schriftgröße)
- `font-style` (Darstellungsstil, normal/kursiv)
- `font-variant` (Kapitälchen, »small-caps«)
- `font-weight` (Fettegrad)

4.3 Das CSS-Boxmodell

Mit **CSS-Box** wird die Metapher umschrieben, die CSS einsetzt, um mit Element-Containern umzugehen – sie bestimmt den Raum, den das Element auf der Seite einnimmt. Angewendet wurde dieses Prinzip bereits auf den Container `#inhalt` und teilweise auf die Überschrift `<h1>`. Hier soll es ausführlicher beschrieben werden.

Vollständig anwendbar ist das Boxmodell nur auf Blockelemente. Mit Hilfe der Box-Properties können beliebige Abmessungen und Präsentationseigenschaften zugewiesen werden.

Als **Box-Properties** können gesetzt werden (siehe Abbildung 4.7):

- **Innenbreite** (`width`-Property)

  Die Breite des Bereichs für den Textinhalt
- **Innenhöhe** (`height`-Property)

  Die Höhe des Bereichs für den Textinhalt
- **Innenabstand** (`padding`-Property)

  Der Abstand zwischen Textinhalt und Rahmen
- **Rahmeneigenschaften** (`border`-Property)

  Breite, Farbe und Erscheinungsbild des Rahmens
- **Außenabstand** (`margin`-Property)

  Abstand des Elements von benachbarten oder übergeordneten Containern

Hierbei kann eine Eigenschaft pauschal für alle vier Seiten des Containers (`top`, `right`, `bottom`, `left`) oder für eine einzelne Seite gesetzt werden.

Man kann also gezielt ein Margin für alle vier Seiten zuweisen:

```
margin:2em;
```

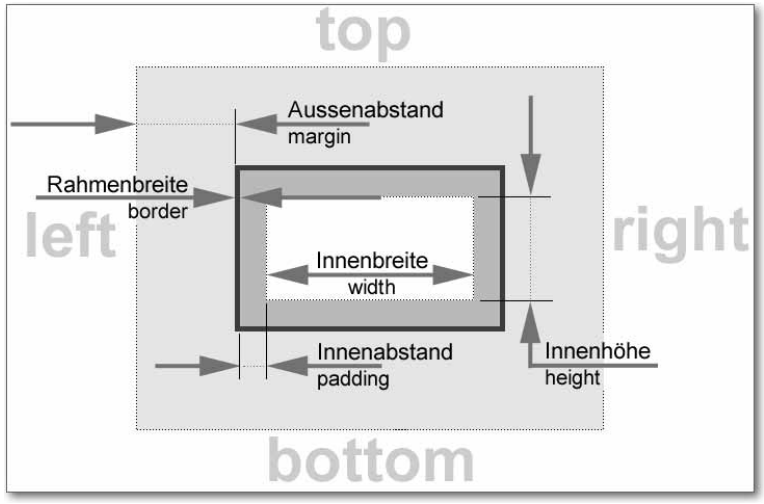

Abbildung 4.7 Die CSS-Box mit Abmessungen

Oder dies auf eine Containerseite beschränken – hier die linke:

```
margin-left:15em;
```

Analog geht auch `margin-top`, `margin-right`, `margin-bottom`. Die gleiche Methode gilt auch für die anderen Box-Properties. Da Einzelzuweisungen für mehrere Seiten etwas Mühe machen, gilt auch folgende Abkürzung:

property: *oben rechts unten links*;

So können verschiedene `margin`-Werte gleichzeitig gesetzt werden:

```
margin:0 1em 1em 15em;
```

Dies nullt den Abstand oben, setzt den für rechts und unten auf 1 `em` und den linken Rand auf 15 `em`. Für den Wert 0 braucht keine Einheit angegeben werden.

Properties für Farben und Hintergründe

Diese Eigenschaften sind nicht nur auf die CSS-Box, sondern auf alle Elemente mit einer sichtbaren Oberfläche anwendbar, beschränken sich daher nicht auf solche mit Textinhalt. Lediglich die `color`-Property bezieht sich auf eine Schrifteigenschaft. In diesem Kapitel wurde dem `<body>`-Element ein Hintergrundbild zugewiesen, desgleichen der Überschrift `<h1>`. Auch der Container `#inhalt` erhielt eine Hintergrundfarbe.

Hier eine Auswahl der wichtigsten Farb- und Hintergrund-Properties:

- `background` (Sammel-Property für Hintergrundeigenschaften)
- `background-attachment` (Hintergrund fixiert oder nicht)
- `background-color` (Hintergrundfarbe)
- `background-image` (URL des Hintergrundbildes)
- `background-position` (Position des Hintergrundbildes)
- `background-repeat` (Hintergrundbild wiederholen)
- `color` (Vordergrundfarbe, d.i. Textfarbe)

4.4 Zusammenfassung

Aus diesem Kapitel sollten Sie folgende Aussagen mitnehmen:

- CSS ist genausowenig eine Programmiersprache wie HTML oder XHTML, sondern eine rein deklarative Präsentationssprache In CSS wird nicht programmiert, sondern es werden **Regeln** aufgestellt.
- CSS-Anweisungen definieren die Präsentation des Inhaltes eines **HTML-Elementcontainers**.
- Eine CSS-Anweisung besteht aus einem **Selektor** und einer Reihe von **Eigenschaftsdeklarationen**, wobei der Selektor das zu stylende Element bestimmt und die Deklarationen die Art seiner Präsentation regeln.
- Es können beliebige **Eigenschaften** wie Farbe, Schrift, Abmessungen und Hintergrundeigenschaften zugewiesen werden.
- CSS-Präsentationsvorschriften können aus Quellen mit unterschiedlichen Relationen zum HTML-Dokument **zusammengefasst** werden.
- CSS-Präsentationsvorschriften dürfen sich **addieren** und gegenseitig **widerrufen** (können sich also überschreiben). Die Methode, nach der hierfür ihre Rangfolge bestimmt wird, bezeichnet man als **Kaskadierung**.
- **Kommentare** in Stylesheets erfolgen in der Form `/* Kommentar */`. Sie dürfen nicht verschachtelt werden.
- Die **CSS-Box** stellt eine Metapher für die Bemessung eines Blockelements mit Innenbreite, Innenabstand, Rahmen und Außenabstand dar. Sie dient ebenfalls zur Zuweisung von Hintergrundeigenschaften.

Wie geht es weiter?

Das folgende Kapitel beschreibt die Grundlagen eines auf CSS und `<div>`-Containern basierenden Seitenlayouts und zeigt hierfür verschiedene Möglichkeiten auf. Außerdem werden der Aufbau der Seitenoberfläche, ihre funktionalen Bereiche und mit diesen zusammenhängende Aspekte erläutert.

In diesem Kapitel werden verschiedene Ansätze des Seitenlayouts mit XHTML und CSS demonstriert. Hierbei wird auf der im vorangegangenen Kapitel erstellten Beispielseite aufgebaut.

5 Seitenlayout mit CSS

Zu den Präsentationsaufgaben, die CSS für HTML-Seiten übernimmt, zählt auch das gesamte Seitenlayout. Dies, wie es früher üblich war, auf Layouttabellen zu basieren gilt zu Recht als unmodern. Unabhängig davon, wie ein Layout erreicht wird, gelten jedoch einige Grundüberlegungen.

5.1 Grundüberlegungen zum Seitenaufbau

Eigentlich lässt sich die Mehrzahl der Grundlayouts von Webseiten auf die gleichen **vier Grundbereiche** reduzieren, die allerdings in sich gelegentlich noch feiner unterteilt werden. Auch die Positionierung der Bereiche im Browserfenster folgt einem Grundmuster:

- Oben liegender Kopfbereich
- Navigationsbereich links unterhalb des Kopfbereichs
- Inhaltsbereich rechts neben dem Navigationsbereich
- Fußbereich unterhalb des Inhaltsbereichs (unterer Seitenrand)

Ein Grund für diese Konstanz liegt in der üblichen **Lese- und Blickrichtung** des Nutzers (zumindest was westliche Länder mit Leserichtung von links nach rechts angeht) sowie in der Tatsache, dass die Positionen der Bildschirminhalte aus technischen Gründen in der linken oberen Ecke ansetzen.

Eine ebenso gebräuchliche Aufteilung weicht insofern ab, als dass sie den Navigationsbereich quer unterhalb des Kopfbereiches anordnet. Im vorliegenden Beispiel (**www.spiegel-online.de**) enthält der Kopfbereich zusätzlich globale Navigations- und Suchfunktionen.

Der Inhaltsbereich ist zweispaltig ausgelegt mit einer links angeordneten Hautspalte und einer rechts von dieser liegenden Kontextspalte. Der Fußbereich

befindet sich am Seitenende und ist nur durch Scrollen erreichbar. In diesem Fall enthält er Kontaktinformationen und Navigationsmöglichkeiten im Sinne einer Hilfsnavigation.

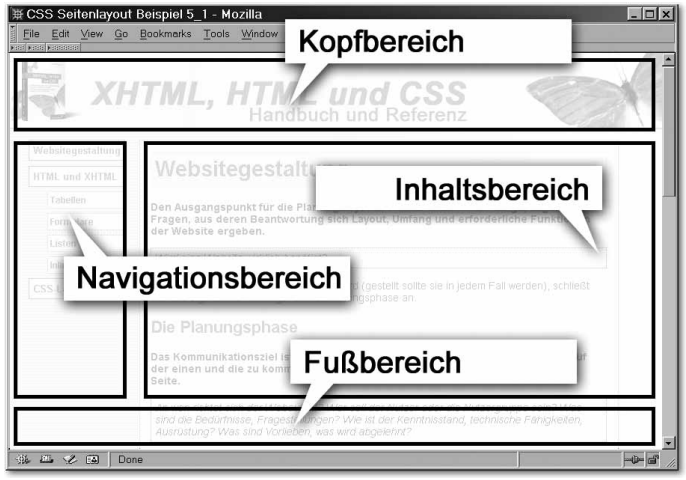

Abbildung 5.1 Grundaufteilung des Fensters in Layoutbereiche

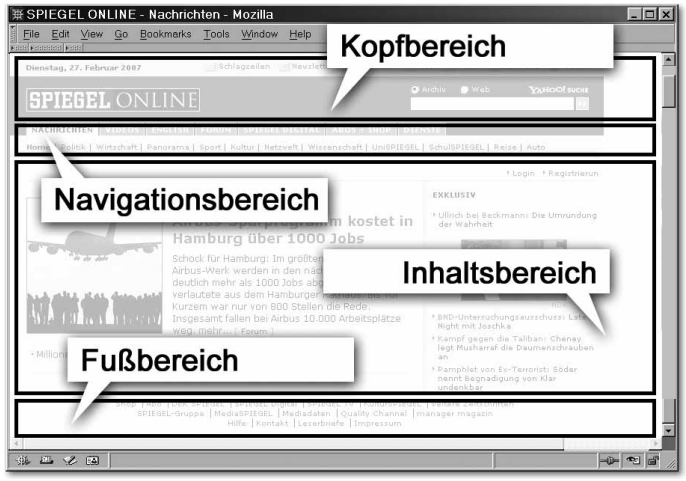

Abbildung 5.2 Alternative Aufteilung des Fensters in Layoutbereiche

Welches dieser Muster man bevorzugt, ist Geschmackssache bzw. hängt von Thema und Zielgruppe der Webseite ab. Entscheidend dürfte die Menge der auf den ersten Blick zu vermittelnden Information sein, die den potentiellen Nutzer zum Bleiben veranlasst.

In zweiter Linie entscheidet die **Zahl der darzustellenden Menüpunkte** über Abmessungen und somit Position der Navigation. Dropdown-Menüs haben den Vorteil, Platz zu sparen – andererseits sind die Navigationsmöglichkeiten nicht alle auf Anhieb ersichtlich.

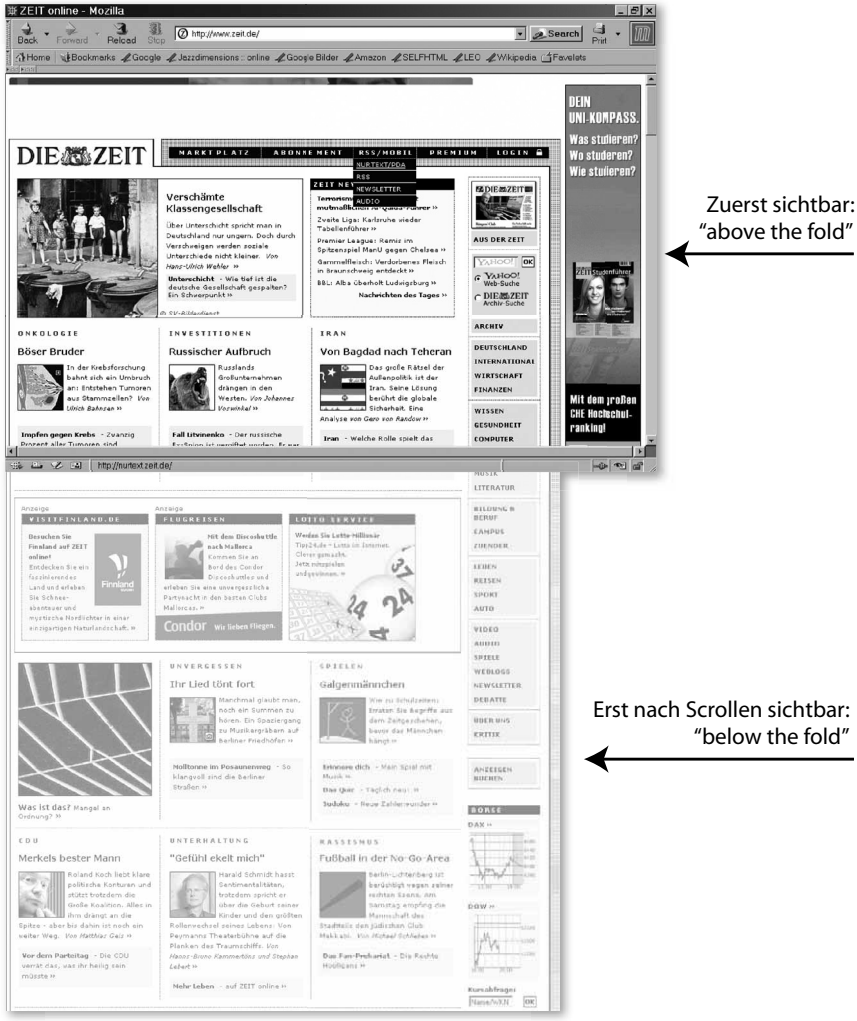

Abbildung 5.3 Die wichtigsten Inhalte müssen sofort sichtbar sein.

Prinzip 1 – Wichtiges zuerst

Aus dem amerikanischen Zeitungswesen stammen die Begriffe »above the fold« und »below the fold«. Ursprünglich bezeichnen sie die bei der Präsentation beim Zeitschriftenhändler sichtbare obere Hälfte einer Zeitung, die das Käuferinteresse

wecken muss. Die Metapher ist ebenso gut auf Webseiten anwendbar, wo man mit »above the fold« jenen Teil der Seite meint, der nach dem Laden sichtbar ist, ohne dass der Besucher zum Scrollen gezwungen wird.[1]

Alles, was wichtig ist (Hauptinfo, Eyecatcher, die Navigation, das Thema der Website) muss ohne Scrollen sofort zu sehen und zu erkennen sein. Weniger wichtige Themen oder solche, die von Besuchern aktiv aufgesucht werden, können weiter unten angeordnet werden (dementsprechend »below the fold«).

Als selbstverständlich kann man aus diesem Prinzip folgern, dass zu den Teilen der Webseite, die sofort zu sehen sein müssen, die globale Navigation gehört, aber auch die Hauptnavigation. Sollte diese in der Vertikalen aus dem sichtbaren Bereich herausragen, so müssen die relevanten Navigationspunkte oben angeordnet sein. Um den Nutzer nicht gleichzeitig zu überfordern, gilt als Faustregel, vertikal nicht mehr als sieben Hauptnavigationspunkte einzusetzen.

Prinzip 2 – Konsistenz bieten

Eine Website sollte konsistent sein, d.h. ein durchgängig gleiches »Look and Feel« bieten. Navigation, Gestaltung werden dadurch vertrauenerweckend. Bereiche können durch verschiedene Leitfarben unterschieden werden.

Ein (nach meiner persönlichen Meinung) gutes Beispiel hierfür bietet die Website **www.spiegel-online.de**, die bei sehr konsistentem Layout die einzelnen Rubriken durch Leitfarben kennzeichnet. Trotz einiger Flexibilität in der Aufteilung des Inhaltsbereichs bleiben wesentliche funktionale Bestandteile wie Navigation und Suche stets an Ort und Stelle. Alle essentiellen Inhalte sind »above the fold« angeordnet.

Natürlich ist Konsistenz nicht das Maß aller Dinge und kann übertrieben werden. Dies ist dann der Fall, wenn gleichartige Bedeutung oder Handhabung optisch suggeriert werden, wo dies in Wirklichkeit gar nicht der Fall ist: Es ist genauso wichtig, visuell »inkonsistent« zu sein, wenn Unterschiede oder ein Wechsel hervorgehoben sollen, wie es wichtig ist, visuell »konsistent« zu bleiben, wenn Gleichartigkeit oder Konstanz vorliegen.

1 Nach Roger Black, *Websites that work*, Adobe Press 1997.

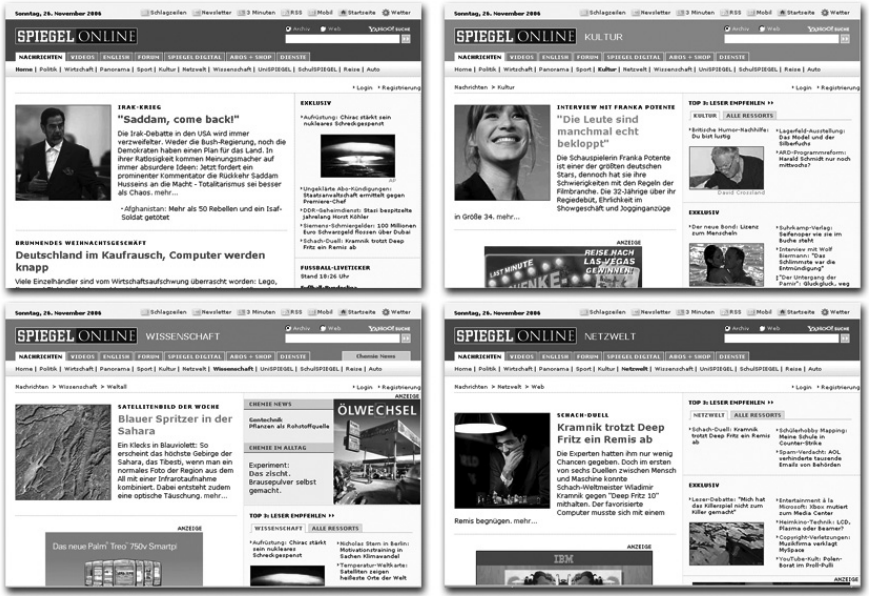

Abbildung 5.4 www.spiegel-online.de – Beispiel für Konsistenz im Design

Prinzip 3 – Es gibt kein Prinzip

Keine Regel ohne Ausnahme – sobald die Botschaft oder Zielgruppe dies erfordert, kann gegen jedes beliebige Gestaltungsprinzip verstoßen werden. Ein interessantes Beispiel dafür, dass auch horizontales Scrollen als Grundlage eines Designs dienen kann, ist die als Galeriemetapher ausgeführte Unterwebsite von **www.zeit.de**, die den User auf einem virtuellen Spaziergang von einem Ausstellungsraum zum nächsten führt (**http://apollo.zeit.de/galerie/**). In diesem Fall liegt der Navigationsbereich unterhalb des Inhaltsbereichs – der Kopfbereich (hier nur rudimentär) befindet sich am gewohnten Ort.

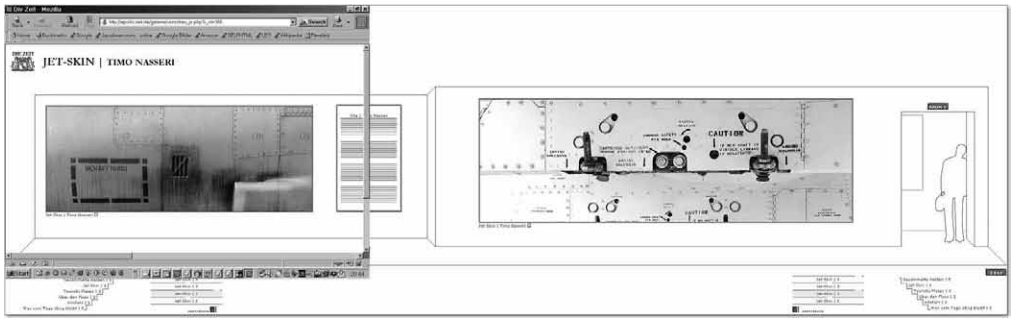

Abbildung 5.5 Horizontales Scrollen ist nicht immer schlecht (www.zeit.de).

Bei Webseiten, deren Botschaft bereits in der Gestaltung selbst liegt und nicht in der Übermittlung von Informationen, muss der Webdesigner sich von gängigen Vorgaben lösen. Dies kann soweit gehen, dass (wie im vorliegenden Falle) auch gegen Prinzipien der Barrierefreiheit verstoßen werden darf.

5.2 Umsetzung der Bereiche in Quellcode – Floatlayout

Aus den Vorüberlegungen lässt sich ableiten, dass man, egal welches Layout man auch bevorzugt, sich zumindest für eine seitenübergreifend konstante Gestaltung entscheiden sollte. Das Quellcodegerüst der Seite muss aber, um überall gleich bleiben zu können, in ein geometrisch stabiles Layout überführt werden, um wechselnde Inhalte aufzunehmen.

Am Anfang steht der -Container

Im letzten Kapitel wurde gezeigt, wie ein `<div>`-Container auf der Grundlage des CSS-Box-Modells mit Abmessungen versehen werden kann. Daraus ließe sich schließen, dass ein Layout mit vier Grundbereichen nicht mehr als vier `<div>`-Container benötigen wird. Der erste Ansatz könnte im Quelltext daher wie folgt sein (Dokumentkopf weggelassen):

```
<body>
    <div id="kopfbereich"><h3>Kopfbereich</h3></div>
    <div id="navigation"><h3>Navigation</h3></div>
    <div id="inhalt"><h3>Inhaltsbereich</h3></div>
    <div id="seitenfuss"><h3>Seitenfuß</h3></div>
</body>
```

Listing 5.1 Quelltextausriss aus beispiel1.html

Die `<div>`-Container sind hier zur optischen Hervorhebung mit einem Rahmen, Margin und Padding versehen. Die CSS-Anweisungen lauten wie folgt:

```
div  { padding:1em;
       margin-top:8px;
       border:2px solid #000; }
```

Die Darstellung im Browserfenster folgt der Quelltextreihenfolge, was nicht verwunderlich ist, da noch keine Maßnahmen getroffen wurden, um die Bereiche zu positionieren. (Die Top-Margin dient dazu, die Container optisch etwas zu separieren. Wenn sie einander berühren sollen, lassen Sie sie weg.)

Abbildung 5.6 Die vier Basiscontainer – ordentlich gestapelt

Schritt 1 – Die Containerbreite setzen

Da Navigation und Inhalt nebeneinander stehen sollen, besteht die erste Maßnahme darin, die Breiten der beiden Container entsprechend zu manipulieren. Um etwas Luft zu haben (den Containern wurde der Optik halber ein Margin gegeben, das später wieder zu beseitigen sein wird), wird der Navigationscontainer auf 25 %, der Inhaltscontainer auf 70 % seiner normalen Breite gesetzt. Die beiden anderen Container werden auf 95 % gesetzt (*Hinweis:* Würden Sie 100 % nehmen, so bekämen Sie bei diesem Szenario horizontale Scrollbalken).

Es existieren jetzt also Definitionen für alle vier Bereiche. Zur Verdeutlichung erhielten Navigation und Inhalt jeweils eine Hintergrundfarbe (Abbildung 5.7):

```
#kopfbereich  { width:95%; }

#navigation   { width:20%;
                background-color:#ff5500; }

#inhalt       { width:70%;
                background-color:#ffff00; }

#seitenfuss   { width:95%; }
```

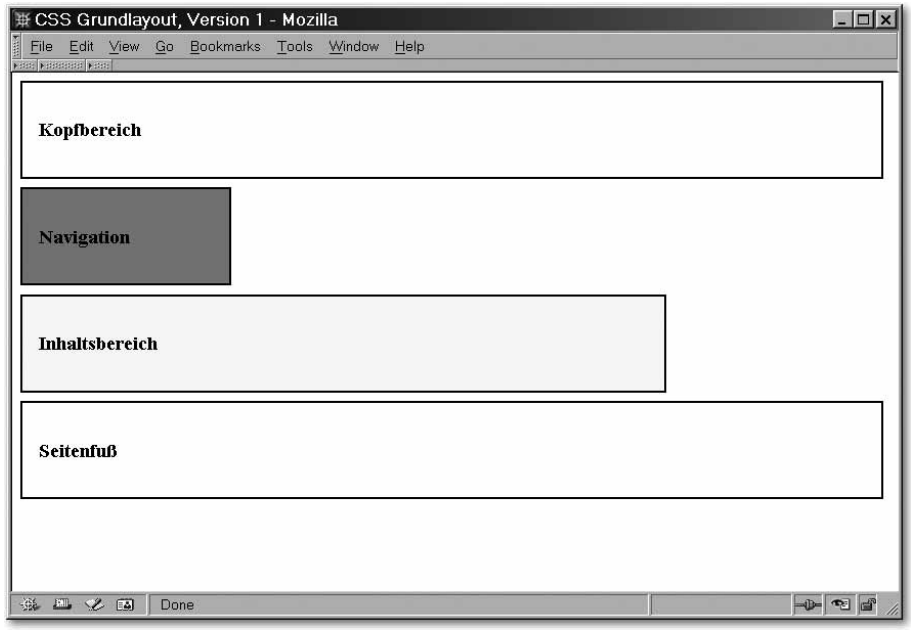

Abbildung 5.7 Die Basiscontainer nach der Breitenzuweisung

Schritt 2 – Die Navigation floaten

Als nächster Schritt wird die Navigation nach links »gefloatet«.

```
#navigation   { width:20%;
                background-color:#ff5500;
                float:left; }
```

Floaten bewirkt, dass der Container aus dem Zeilenkontext entfernt wird. Man kann ein Objekt entweder nach links (wie hier) oder nach rechts floaten. Es wird an den entsprechenden Rand des Browserfensters (genauer: des umliegenden Containers) bewegt. In diesem Fall bleibt die Position des Navigationscontainers gleich, da er sich ohnehin bereits am linken Rand befindet. Was sich jedoch zusätzlich ändert, ist das **Verhalten der folgenden Container**, da für sie das gefloatete Objekt verschwindet.

Für den Inhaltscontainer macht dies einen gewaltigen Unterschied, da er (aus seiner Sicht) sich nun direkt nach dem Kopfcontainer eingliedern kann. Allerdings schiebt er sich hierdurch hinter den Navigationscontainer, wird von diesem daher teilweise verdeckt (Abbildung 5.8).

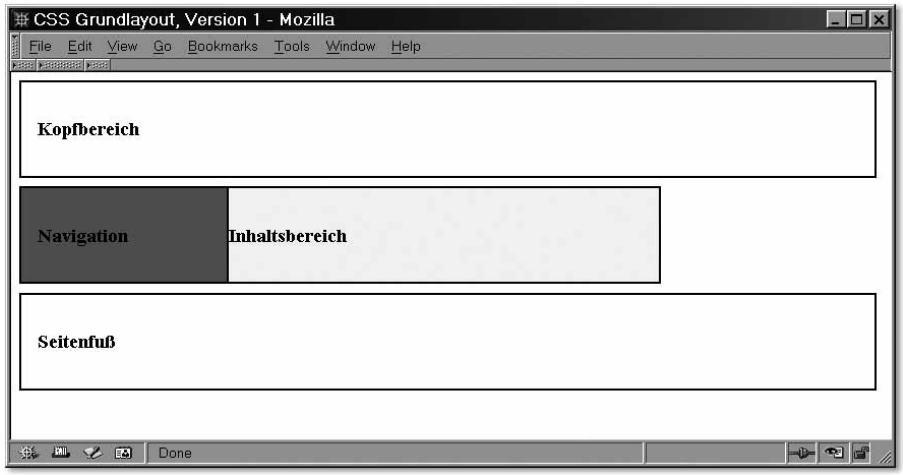

Abbildung 5.8 Nach dem Float sitzt der Inhalt hinter der Navigation.

Schritt 3 – Den Inhalt zurechtrücken

Mittels eines seitlichen Margins wird der Inhaltscontainer nun soweit nach rechts gerückt, dass er nicht mehr verdeckt wird:

```
#inhalt  { width:70%; background-color:#ffff00;
           margin-left:25%; }
```

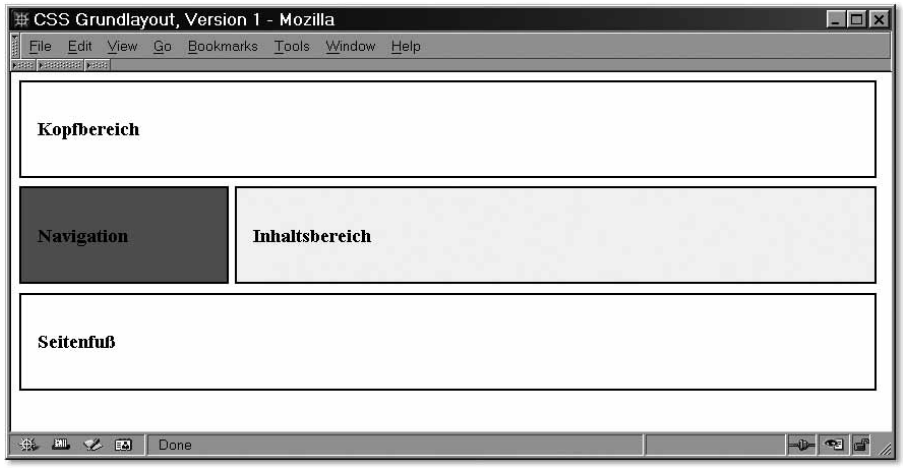

Abbildung 5.9 Ein linkes Margin für den Inhalt rückt diesen zurecht.

Schritt 4 – Den Seitenfuß clearen

Um den Sinn des nächsten Schrittes zu verstehen, muss ein kleines Experiment vorgenommen und der gefloatete Navigationscontainer auf eine größere Höhe gesetzt werden. Das Problem taucht auf, sobald die Navigation höher ist als der Inhaltsbereich, weil in diesem zuwenig Inhalt steht. Ein Wert von 200 px genügt zur Demonstration:

```
#navigation { width:20%; background-color:#ff5500;float:left;
height:200px; }
```

Plötzlich taucht das gleiche Problem, das vorher mit dem Inhaltscontainer auftrat, an neuer Stelle auf – der nun zu lange Navigationscontainer überlagert den Seitenfußbereich, für den man hier selbstverständlich nicht mit `margin-top` arbeiten kann (dies wäre nur möglich, handelte es sich um eine konstante Abweichung – die Höhe der Container kann jedoch je nach Inhalten wechseln).

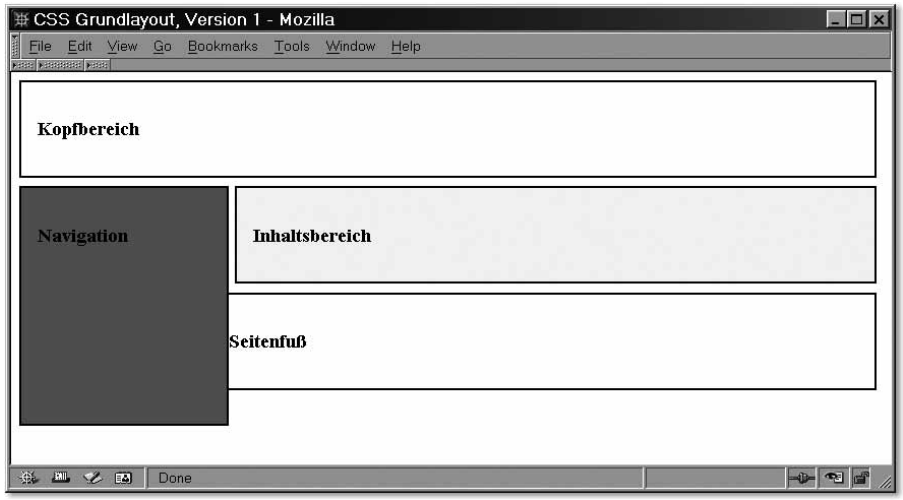

Abbildung 5.10 Die zu hohe Navigation schiebt sich über den Seitenfuß.

Die Lösung besteht darin, das Verhalten des Seitenfußes gegenüber dem gefloateten Objekt zu ändern. Normalerweise wird er versuchen, sich an das Vorgängerobjekt anzugliedern (hier den Inhalt) und das gefloatete Objekt zu ignorieren bzw. zu umfließen. Dieses Verhalten wird durch Setzen des CSS-Properties `clear` beendet – in diesem Fall gegenüber nach links gefloateten Objekten:

```
#seitenfuss { width:95%; clear:left; }
```

Der Seitenfuß rückt nun an die Randposition nach dem gefloateten Objekt (siehe Abbildung 5.11). Gegenüber einem nach rechts gefloateten Objekt verwendet

man `clear:right` (dies wäre in diesem Fall wirkungslos). Eine gute Alternative, die gleichzeitig gegenüber nach rechts gefloateten Objekten wirkt, ist `clear:both`.

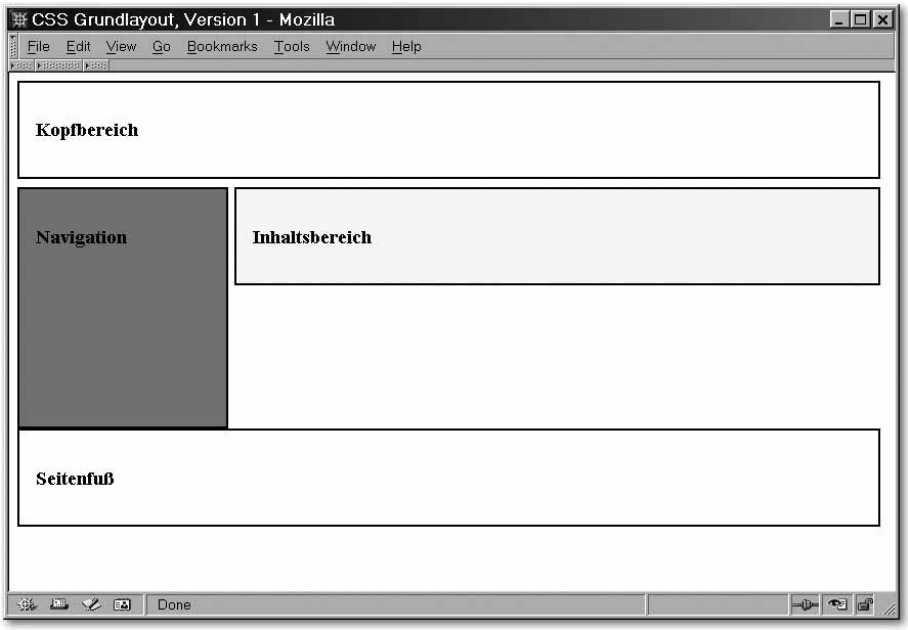

Abbildung 5.11 Die clear-Property beseitigt den Missstand.

5.3 Quelltextreihenfolge und Accessibility

An dieser Stelle ist eine Zwischenüberlegung angebracht. Die Dokumentcontainer wie eben demonstriert zu positionieren, ist die eine Sache. Ein Nutzer, der die Seite wie vorgesehen betrachtet, sieht alles an der gewünschten Position. In der Praxis sind die Container allerdings nicht leer, sondern enthalten viel oder sogar sehr viel Inhalt. Erinnern Sie sich daran, dass der Navigationscontainer im Quelltext vor dem eigentlichen Inhalt steht (siehe Abbildung 5.12, links).

5.3.1 Die Wahl der Quelltextreihenfolge

Für User Agents, die Abmessungen und Positionen der Objekte verarbeiten, stellt dies kein Problem dar. Andere User Agents, die das Dokument in Quelltextreihenfolge darstellen oder direkt den Quelltext verarbeiten, müssen sich zunächst durch eine umfangreiche Navigation durcharbeiten. Zu diesen User Agents gehören beispielsweise Screenreader, Mobilgeräte (PDAs, Mobiltelefone) und auch

Suchmaschinen. Wir bewegen uns mit diesen Überlegungen daher wieder in das Gebiet der Barrierefreiheit.

Hier gibt es zwei Ansätze, nach denen vorgegangen werden kann:

- Änderung der Quelltextreihenfolge (Inhalt vor Navigation)
- **Sprunglinks am Dokumentanfang** zu Inhalt und Navigation

Die gewünschte Wirkung besteht in der Erleichterung des Zugangs zu den Inhalten für bestimmte Nutzergruppen. Die **Sprunglinks** (auch »Skiplinks«) werden sinnvollerweise im Kopfbereich platziert, so dass ein Nutzer mit Screenreader oder mit Handybrowser diese zuerst sieht und, sofern er möchte, direkt zum Seiteninhalt springen kann (schematisch in Abbildung 5.12).

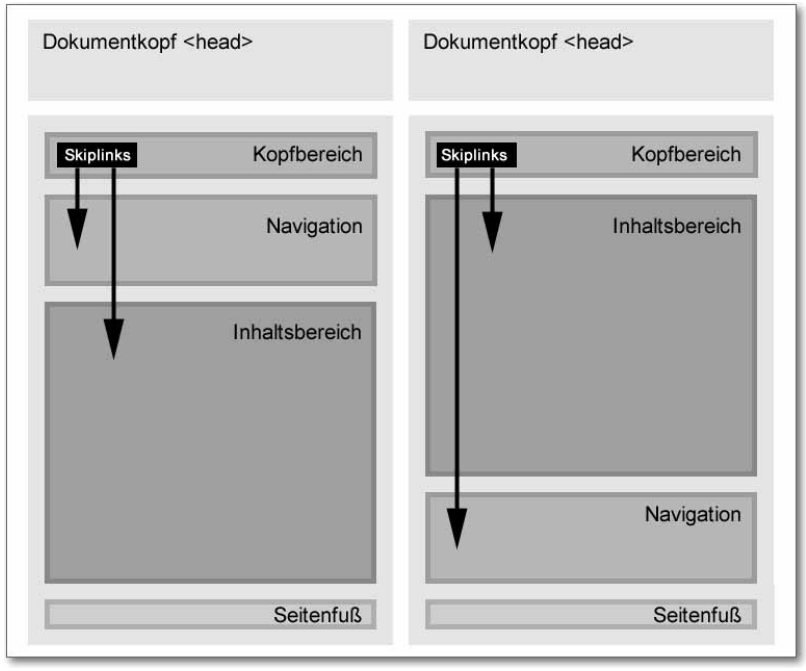

Abbildung 5.12 Normale und reorganisierte Quelltextreihenfolge

Der Vorteil von Sprunglinks ist unabhängig von der eventuellen Reorganisation des Quelltextes, also der Platzierung der Navigation nach dem Inhalt. Zur Erinnerung: Da die Präsentation, also auch die Positionierung auf dem Bildschirm durch Stylesheets geschieht, sehen Nutzer mit abgeschalteten Stylesheets oder solche, die Stylesheets nicht interpretieren, das Dokument ohnehin in Quelltextreihenfolge.

Den Inhalt zuerst zu bringen (siehe Abbildung 5.12, rechts), ist eine Überlegung wert – für Suchmaschinen mag dies günstiger sein, aber auch für Nutzer, die in erster Linie an den Informationen interessiert sind und es missbilligen, sich auf jeder Seite erneut durch die Navigation zu bewegen (sofern sie nicht mit einem normalen Webbrowser auf die Site gehen).

5.3.2 Einbau der Sprunglinks

Als kleiner Zwischenexkurs soll hier eine Möglichkeit des Einbaus von Sprunglinks gezeigt werden (es gibt wie immer auch Alternativen).

Die Ziele der Sprunglinks

Bevor der eigentliche Link erstellt wird, muss geklärt sein, wohin die Links eigentlich zielen. Bisher kamen Hyperlinks nur in Zusammenhang mit Verknüpfungen zu externen Dokumenten vor – hier geht es um einen dokumentinternen Link. Praktischerweise sind die beiden Ziele, der Navigationscontainer und der Inhaltscontainer, bereits eindeutig gekennzeichnet: Beide Container besitzen einen ID:

```
<div id="navigation"> ... </div>
<div id="inhalt"> ... </div>
```

Dieser ID macht die Container als Linkziele erreichbar – als Hyperreferenz muss nur der Identifiername angegeben werden.

Die Sprunglinks

Die eigentlichen Links sind einfach zu erstellen. Üblicherweise werden sie in einen `<div>`-Container gestellt, der positioniert wird und ebenfalls einen ID bekommt (auch ein Klassenattribut ist hier möglich):

```
<div id="sprunglinks">
<a href="#navigation">Zur Navigation springen</a> |
<a href="#inhalt">Zum Inhalt springen</a>
</div>
```

Achten Sie darauf, dass zwischen den `<a>`-Containern ein senkrechter Trennstrich steht. Die ist erforderlich, da die Links sonst für Bildschirmlesegeräte »ineinander fließen«, also nicht zu trennen sind. Als Wert des `href`-Attributs wird der ID-Bezeichner des Zielcontainers mit vorangestelltem Hash-Zeichen verwendet.

5.4 Umsetzung der Bereiche in Quellcode – Positionslayout

Entscheidet man sich für eine Umstellung im Quellcode, so kann das Seitenlayout nicht durch Floats erreicht werden, sondern muss mit Positionierungen vorgenommen werden.

Zunächst muss das Dokument umorganisiert werden. Die Sprungmarke wird im **Kopfcontainer** eingesetzt (später gesellen sich noch weitere Inhalte hinzu), der Inhaltscontainer kommt vor der Navigation zu stehen. Die Position des Seitenfußes bleibt:

```
<body>
  <div id="kopfbereich">
   <div id="sprunglinks">
   <a href="#navigation">Zur Navigation springen</a> |
   <a href="#inhalt">Zum Inhalt springen</a>
   </div>
</div>
<div id="navigation"> ... </div>
<div id="inhalt"> ... </div>
<div id="seitenfuss"> ...</div>
</body>
```

Listing 5.2 Der Sprunglink im Dokument

5.4.1 Schritt 1 – Festlegung der Containerbreiten

Im ersten Schritt werden wieder die Containerbreiten festgelegt, wobei dieselben Abmessungen wie beim Floatlayout verwendet werden. Der Kopfcontainer bekommt jedoch eine feste Höhe in Pixel zugewiesen. Dies wird man in der Regel ohnehin vornehmen, da dort als Inhalte vorgesehene Logos und Stimmungsbilder ohnehin festgelegte Abmessungen besitzen.

```
#sprunglinks { margin:0; padding:5px;
               border:1px solid #000; }
```

[»] *Hinweis:* Die Sprunglinks bekommen nur deshalb eine Darstellungsanweisung, um die allgemeine Anweisung für `<div>`-Container zu überschreiben, welche die Rahmen für die Screenshots bewirkt. Die restlichen Angaben entsprechend noch denen beim Floatlayout:

```
#kopfbereich { width:95%; height:65px; }
#navigation  { width:20%; background-color:#ff5500; }
#inhalt      { width:70%; background-color:#ffff00; }
#seitenfuss  { width:95%; }
```

Das `clear`-Property des Seitenfußes wurde entfernt, da sie nur im Floatkontext einsetzbar ist.

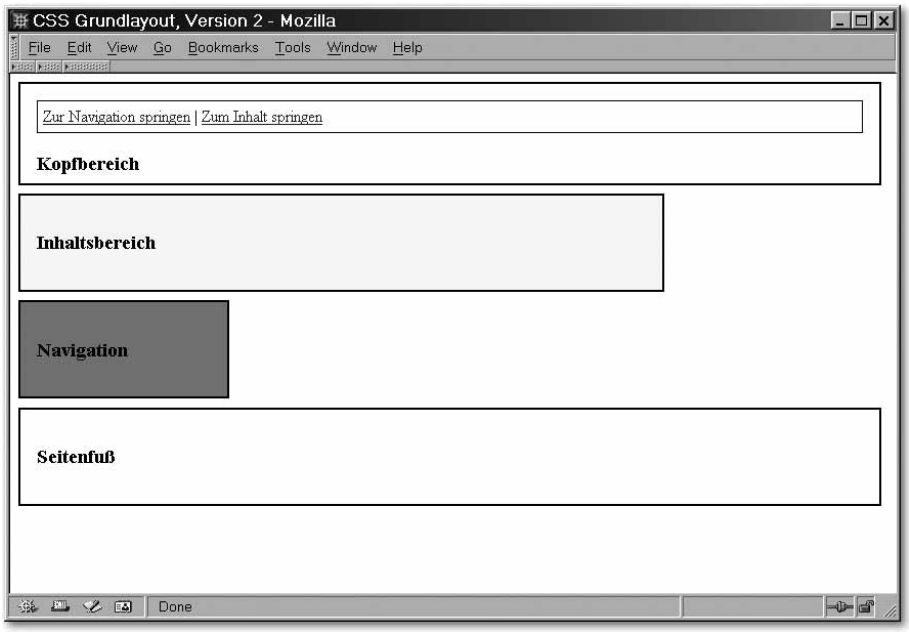

Abbildung 5.13 Die Container sind dimensioniert.

5.4.2 Schritt 2 – Platz für die Navigation schaffen

Derzeit (Abbildung 5.14) steht der Inhaltscontainer noch an der Position, die durch die Navigation eingenommen werden soll. Analog zum Floatlayout wird er einfach durch eine `margin`-Angabe nach rechts verschoben, so dass nun der erforderliche Platz frei gemacht ist.

```
#inhalt  { width:70%;
           background-color:#ffff00;
           margin-left:25%; }
```

Der Navigationscontainer sitzt noch unbeeindruckt unterhalb des Inhaltes. Er wird im nächsten Schritt nach oben gerückt, wie in Abbildung 5.14 angedeutet.

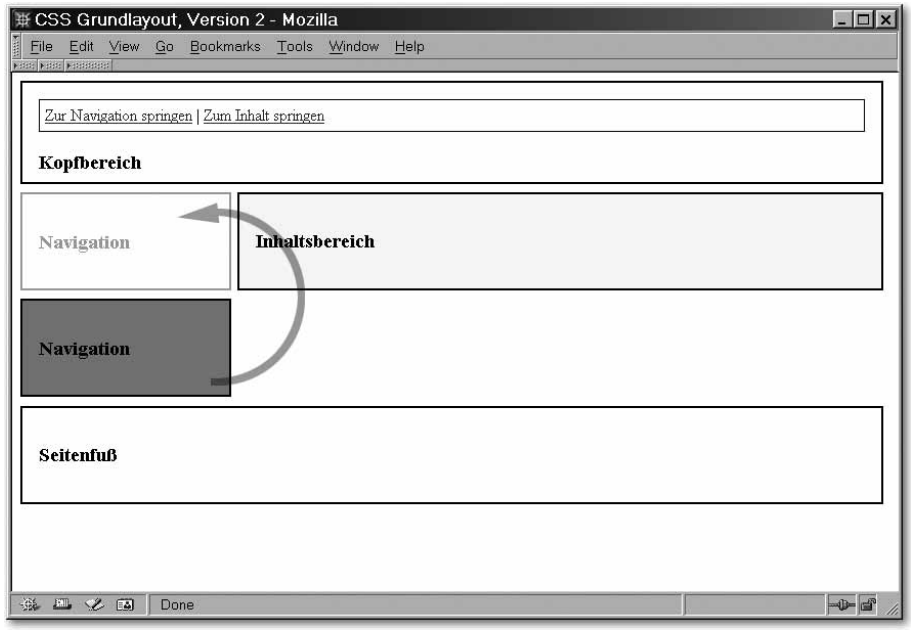

Abbildung 5.14 Der Inhaltscontainer wird aus dem Weg geschafft.

5.4.3 Schritt 3 – Positionierung des Navigationscontainers

Der Navigationscontainer wird nun *absolut* positioniert. Mit »absolut« ist hier gemeint, dass der Bezugspunkt für Position gegenüber derjenigen des unmittelbar übergeordneten Containers zu verstehen ist – in diesem Fall ist dies das `<body>`-Element, was aber nicht immer der Fall zu sein braucht.

Der »Nullpunkt« des **Bezugskoordinatensystems** ist hier also die linke obere Ecke des Browserfensters. Von hier ab wird eine `left`-Koordinate nach rechts gerechnet und eine `top`-Koordinate nach unten (auch die Richtung nach unten wird als positive Größe ausgedrückt).

Hier wird die Navigation 109 Pixel nach unten verschoben. Dies geschieht mit der `top`-Angabe. Die `left`-Koordinate wird nicht genannt, da hier die statische Position des Containers verwendet werden kann:

```
#navigation { width:20%; background-color:#ff5500;
  position:absolute;
  top:109px; }

#inhalt     { width:70%; background-color:#ffff00;  margin-left:25%;}
```

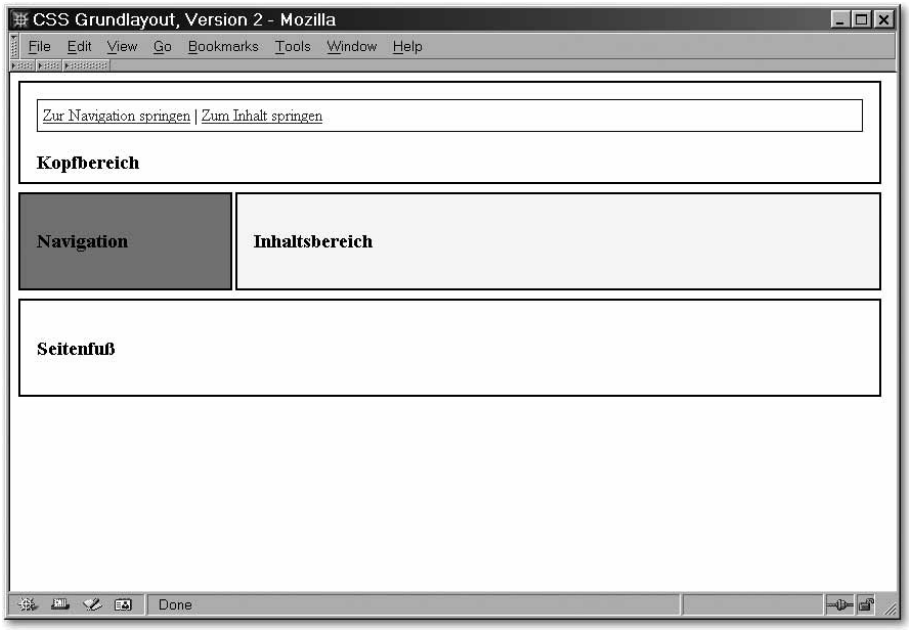

Abbildung 5.15 Der Navigationscontainer ist positioniert.

Das Ergebnis ist durchaus zufrieden stellend. Tritt in diesem Fall jedoch die Problematik mit einem zu großen Navigations- und einem zu kleinen Inhaltscontainer auf, so hilft beim Fußcontainer kein `clear`-Property: Die Navigation überlagert den Container in diesem Fall (Abbildung 5.16). Die einzig mögliche Abhilfe besteht darin zu verhindern, dass der Inhaltsbereich zu kurz wird – hierfür kann man ihm eine Mindesthöhe zuweisen.

Absolut positionierte Elemente sind aus dem Kontext genommen

Sobald ein Element absolut positioniert ist, ist es aus der Dokumentreihenfolge genommen, für im Quelltext nachfolgende Elemente also nicht mehr existent. Dies ist anders als im Float-Kontext, wo durchaus eine Wechselwirkung vorhanden bleibt. Absolut positionierte Elemente »schweben« quasi an ihrer Position über der Seite. Die folgenden Elemente ordnen sich an der nächsten freien Position ein, werden also gegebenenfalls überdeckt.

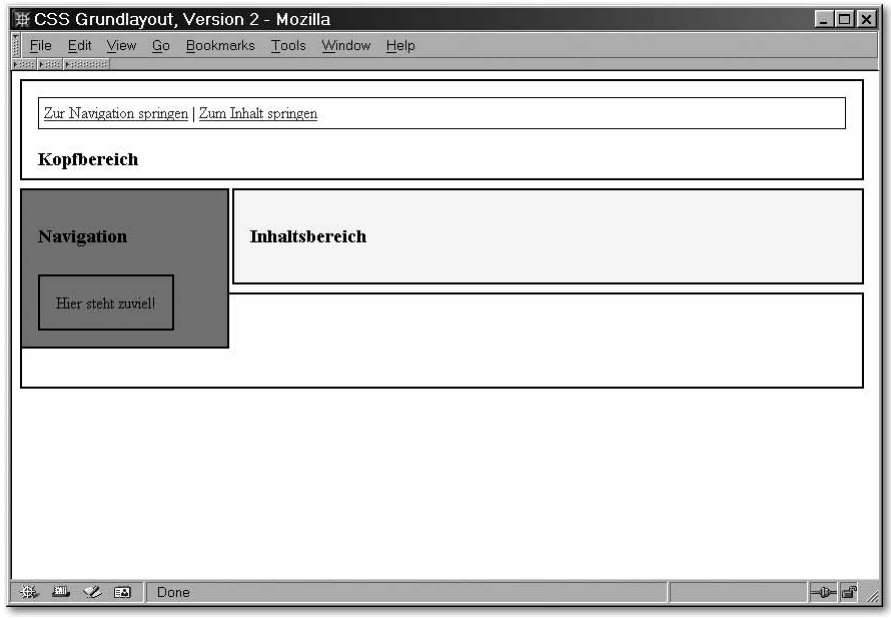

Abbildung 5.16 Der Navigationscontainer überlagert den Seitenfuß.

5.4.4 Absolute und relative Positionierung

Von den vier möglichen Positionierungen sollen hier nur zwei angesprochen werden, nämlich »absolute« und »relative«. Die statische Positionierung (»static«) entspricht der normalen Positionierung ohne weitere Angaben, feste Positionierung (»fixed«) entspricht hinreichend der absoluten Positionierung (nur relativ zum Browserfenster, nicht zum Seitenursprung), so dass eine genauere Erörterung dieser Positionierungsmethoden unterbleiben kann.

Absolute Positionierung

Diese Positionierung bezieht sich, ausgehend von positionierten Objekt stets auf die **Ursprungskoordinaten** (Ecke links oben) des nächst **übergeordneten Blockelements**. Für den Container `id="aussen"` ist dies der `<body>`-Container.

Dessen Ursprungskoordinaten sind `top:0px` und `left:0px`. Zu den Ursprungskoordinaten werden die übergebenen Positionsangaben addiert. Der äußere `<div>`-Container wird daher an `top:50px` und `left:50px` positioniert.

Der Container `id="innen"` bezieht sich bei absoluter Positionierung *nicht* auf den `<body>`, sondern auf den umliegenden `<div>`-Container `id="aussen"`. Dessen Ursprung liegt bei `top:50px` und `left:50px`. Der innere Container kommt in der Summe daher bei `top:100px` und `left:100px` zu liegen (Abbildung 5.17).

```
<body>
  <div id="aussen">
    <h3>au&szlig;en</h3>
    <div id="innen">
      <h3>innen</h3>
    </div>
  </div>
<body>
```

… mit folgenden Positionsangaben:

```
#aussen { position:absolute;
          top:50px; left:50px;
          background-color:#ffff00; }

#innen  { position:absolute;
          top:50px; left:50px;
          background-color:#ff5500; }
```

Die Dimensionen des äußeren Containers bemessen sich nur an den ihm verbliebenen Inhalten (die Überschrift). Der innere Container ist durch die absolute Positionierung aus seinem Kontext entfernt und nimmt dort keinen Raum ein.

Relative Positionierung

Relative Positionierung bezieht sich nicht auf den übergeordneten Container, sondern auf die Koordinaten, an denen sich das Objekt ohne Positionierung befände. Es wird durch relative Positionierung also von dieser Position weggerückt, hinterlässt dort aber eine entsprechende Lücke. Es nimmt an der ursprünglichen Position also noch den gleichen Raum ein, als ob es sich dort befände.

In diesem Fall wird ein `<span>`-Container mit `id="relativ"` in einem `<div>`-Container aus seiner Position verschoben. (Die Positionierung des `<div>`-Containers – hier ist sie absolut – spielt dabei keine Rolle.)

```
<body>
  <div id="aussen2">
   <p>Hier steht ein Container
    <span id="relativ">mit Inhalt</span>
   in der Zeile.</p>
  </div>
</body>
```

Folgende Positionsangaben wurden gemacht (siehe Abbildung 5.17):

```
#aussen2  { position:absolute;
            top:200px; left:50px;
            background-color:#ffff00; }

#relativ  { position:relative;
            top:15px; left:15px;
            background-color:#ff5500;
            border:1px solid black; }
```

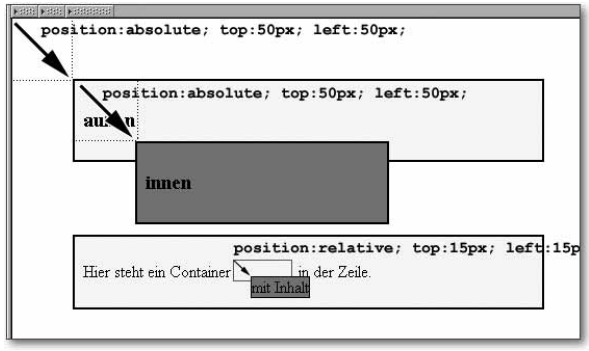

Abbildung 5.17 Absolute (oben) und relative Positionierung im Vergleich

5.4.5 CSS-Eigenschaften für Positionierungszwecke

In CSS existieren eine Reihe von Eigenschaften die im Rahmen der Gestaltung eines Seitenlayouts verwendet werden können. Hier eine Auswahl der wichtigsten (sie werden in Kapitel 19, *CSS-Referenz*, genauer erläutert):

- `clear` (Umbruch nach Float-Element, ja/nein/Seite)
- `float` (Floaten eines Elements, links/rechts)
- `left` (Abstand vom linken Rand)
- `position` (Art der Positionierung)
- `top` (Abstand vom oberen Rand)
- `visibility` (Sichtbarkeit eines Elements)
- `z-index` (Tiefenstaffelung positionierter Elemente)

5.5 Zusammenfassung

Was Sie aus diesem Kapitel mitnehmen sollten:

- Es gibt allgemeine Gestaltungsprinzipien, die man als Webdesigner kennen sollte. Zu diesen zählen: »Wichtiges zuerst« und »Konsistenz wahren«.

- Aufgabenbezogen lässt sich die Oberfläche einer Webseite in mehrere Bereiche unterteilen, die in unterschiedlicher Anordnung stets auftreten.
- Diese Bereiche sind Kopfbereich, Navigationsbereich und Inhaltsbereich sowie der Seitenfuß.
- Eine Grundstruktur kann verhältnismäßig einfach aus `<div>`-Containern aufgebaut werden. Diese müssen anschließend positioniert werden.
- Die Grundtypen eines `<div>`-Layouts sind »Floatlayout« und »Positionslayout«, wobei das erstere mit gefloateten Containern, das zweite mit positionierten Containern arbeitet.
- Das Positionslayout ermöglicht die Reorganisation des Quelltextes, um beispielsweise den Inhaltscontainer vor dem Navigationscontainer anzuordnen, aber optisch anders zu präsentieren.
- Um die Zugänglichkeit der Seite zu verbessern, sollten im Kopfbereich der Seite Sprunglinks, so genannte »Skiplinks« eingesetzt werden, die direkt zum Inhalts- oder Navigationscontainer führen.
- Absolut positionierte Objekte stehen außerhalb des Seitenkontextes. Elemente, die im Seitenfluss verbleiben, werden daher gegebenenfalls überdeckt. Die Position des Objekts wird ab den Ursprungskoordinaten des umgebenden Blockcontainers gerechnet.
- Relativ positionierte Objekte verbleiben im Seitenfluss, nehmen daher dort ihren gewohnten Raum ein. Ihre Positionierung entspricht einer Verschiebung gegenüber der eigentlichen Position.

Wie geht es weiter?

Das folgende Kapitel vertieft die Gestaltung des Inhaltsbereiches und stellt weitere HTML-Elemente vor, die zur Gestaltung von Tabellen, Formularen und Listen benötigt werden. Hierbei wird auch auf deren Gestaltung mit CSS eingegangen und es werden Aspekte der Barrierefreiheit erörtert. Das darauf folgende Kapitel 7, *Navigationsgestaltung mit CSS,* steigt wieder in die Seitengestaltung ein.

Literaturhinweise: Der Hauptschwerpunkt dieses Buchs liegt auf HTML – allein über CSS und Seitenlayout ließen sich bereits dicke Bücher schreiben. An dieser Stelle möchte ich Ihnen daher die hervorragenden Ausführungen von Dirk Jesse und sein Buch *CSS-Layouts* (Galileo Press) empfehlen sowie das Referenzwerk *CSS-Praxis* von Kai Laborenz (ebenfalls Galileo Press), das an Ausführlichkeit nichts zu wünschen übrig lässt.

Dieses Kapitel beschreibt die Verwendung von XHTML-Elementen zur Gestaltung des eigentlichen Dokumentinhalts. Angesprochen werden Überschriften und Textabsätze zur Gliederung, sowie Inlineelemente zur Textgestaltung, aber auch Hyperlinks, Grafiken, Tabellen, Formulare und Listen.

6 (X)HTML im Contentbereich

Nachdem im vorangegangenen Kapitel die Grundlagen eines Dokumentlayouts beschrieben wurden (daran wird im folgenden Kapitel 7, *Navigation*, wieder angeknüpft werden), soll nun näher auf die Gestaltung des eigentlichen Dokumentinhalts und der hierfür zur Verfügung stehenden HTML-Elemente und CSS-Anweisungen eingegangen werden. Thema sind daher Elemente zur Textformatierung und Textauszeichnung, aber auch das Einbinden von Medienobjekten, oder die Erstellung und Gestaltung von Datentabellen und Formularen. Auch die Verwendung von Hyperlinks im Dokument wird vertieft.

6.1 Contentmarkup mit Textabsätzen und Überschriften

Die grundlegenden Elemente zur Inhaltsgestaltung wurden bereits erwähnt – es handelt sich um die, in Kapitel 3, *(X)HTML – die Dokumentstruktur* vorgestellten Blockelemente, wobei zunächst die Überschriften `<h1>` bis `<h6>` und der Textabsatzcontainer `<p>` im Zentrum der Betrachtung stehen sollen. Diese Container gliedern Textinhalte in logische Abschnitte, besitzen allerdings daneben auch Präsentationseigenschaften. Einige Einstellungen, die diese Grundeigenschaften, aber auch allgemeine Textdarstellung betreffen, können bereits auf globaler Ebene erfolgen.

6.1.1 Mögliche globale Grundeinstellungen per CSS

Eine Reihe von HTML-Elementen, darunter auch der Textabsatzcontainer `<p>` sowie alle Überschriftencontainer, besitzen vordefinierte Außenabstände (*margins*), die für einen Abstand zum Vorgänger- oder Folgecontainer sorgen. Sofern Sie diese Grundeigenschaften nicht beibehalten, sondern individuell festlegen möchten, ist es erwägenswert, diese zunächst global auf Null zu setzen.[1] Das glei-

che gilt für Innenabstände (*paddings*), die für einige Elemente in der Darstellung einiger Browser ebenfalls ungleich Null sind.

Nullung der Default-Margins

Für eine globale **Nullung der Default-Margins** der HTML-Präsentation setzen Sie folgende CSS-Anweisung an den Beginn Ihres Stylesheets:

```
* { margin: 0; padding: 0; }
```

Der Stern, der hier als Selektor dient, ist ein Platzhalter für einen beliebigen Typselektor, erfasst also alle Elemente ungeachtet ihres Bezeichners. Man nennt den Stern-Selektor deshalb auch den **universellen Selektor**. Die auf diese Weise erfassten Elemente sind in der folgenden Tabelle aufgelistet.

Beachten Sie, dass Sie nun selbst dafür Sorge tragen müssen, dass im Rahmen des Layouts erforderliche Abstände definiert werden. (Im Gegenzug kommen Ihnen allerdings nicht unvermuteterweise Defaulteigenschaften ins Gehege.) Die **zusätzliche Nullung der Paddings** vereinheitlicht das Verhalten der verschiedenen Browser gegenüber den Elementen – laut den offiziellen Spezifikationen existierten zwar keine Default-Paddings (obwohl sich `<body>` und `<td>` sowie einige weitere Elemente browserspezifisch so verhalten), es wird jedoch in keinem Fall Schaden angerichtet.

Element	Defaultmargin	Element	Defaultmargin
`<body>`	8px	`<p>`	1.12em 0
`<h1>`	0.67em 0	`<blockquote>`	1.12em 40px
`<h2>`	0.75em 0	`<form>`	1.12em 0
`<h3>`	0.83em 0	`<fieldset>`	1.12em 0
`<h4>`	1.12em 0	`<ol>`, `<ul>`	1.12em 0 1.12 em 40px
`<h5>`	1.5em 0 1	`<menu>`, `<dir>`	1.12em 0 1.12 em 40px
`<h6>`	1.67em 0	`<dl>`	1.12em 0 1.12 em 40px

Tabelle 6.1 Defaultmargins gemäß W3C-Spezifikation (tatsächliche Umsetzung variiert)

Globale Grundeinstellungen für den <body>-Container

Einige Grundeinstellungen bezüglich der im Dokument einzusetzenden **Schriften** können im `<body>`-Container global für das Dokument vorgenommen werden. In erster Linie betrifft dies den zu verwendenden **Font**, also die Schriftart. Hierzu dient das Property `font-family`, das Ihnen bereits aus Kapitel 4, *CSS-*

1 Einen radikaleren Ansatz bietet Tantek Çelik in: http://tantek.com/log/2004/undohtml.css.

Grundsyntax, her bekannt ist. Sie können mit dem `color`-Property zusätzlich auch eine **Schriftfarbe** festlegen.

```
body  {
     /* Defaultschriftfarbe: */
        color: #000;
     /* Defaulthintergrundfarbe: */
        background-color: #fff;
     /* Defaultschrift: */
        font-family: arial, helvetica, sans-serif;

     /* verhindert Bug bei relativen Schriftgrößen: */
        font-size: 100.01%;
}
```

Grundsätzlich sollte für Elemente, für die eine Schriftfarbe festgelegt wird, auch eine **Hintergrundfarbe** bestimmt werden (der W3C-CSS-Validator würde das Fehlen einer solchen Definition monieren), wobei auf einen guten Kontrast geachtet werden muss. Hier wurden die Defaultfarben, die der Browser verwendet, explizit gesetzt.

Sofern **Hintergrundgrafiken** eingesetzt werden, gilt entsprechendes für die Lesbarkeit der Schrift vor dem gewählten Hintergrundbild (in diesem Fall sollte dennoch die Definition einer Hintergrundfarbe erfolgen).

Für eine **gleichmäßigere optische Wirkung** des Schriftbildes kann es sinnvoll sein, fett gesetzten Textpassagen (Überschriften oder Betonungen) einen etwas helleren (bzw. dunkleren) Farbton zu geben, um ihnen an Gewicht zu nehmen.

Ist als Defaultschriftfarbe **schwarz** (`#000`) gewählt, so kann hierfür ein dunkles Grau eingesetzt werden.

```
/* Dunkelgrau: */
h1, h2, h3, h4, h5, h6, b, strong { color:#333; }
```

Analog nimmt man bei Defaultschriftfarbe **weiß** (`#fff`) durch Einsatz eines hellen Grautones den fett gesetzten Passagen etwas Härte:

```
/* Hellgrau: */
h1, h2, h3, h4, h5, h6, b, strong { color:#eee; }
```

Last not least kann eine Definition für die Schriftgröße erfolgen, die dann als **Basisschriftgröße** für die Container innerhalb von `<body>` gilt. Hier gibt es zwei Ansätze – Sie können hier eine prozentuale Angabe machen, oder eine in **em**. Im obigen Beispiel ist eine **prozentuale Angabe** gemacht, die die Ausgangsschriftgröße (beinahe) beibehält. Die Definition

```
font-size: 100.01%;
```

beugt Bugs in älteren Browsern (Internet Explorer 5.x und Opera 6.0) vor, die zu Rundungsfehlern bei weiter innen liegenden prozentualen Schriftgrößenangaben führt. Ebenfalls denkbar (ohne jedoch den angesprochenen Bug zu vermeiden) ist eine Angabe in **em**, wobei die Basisschriftgröße auch vergrößert oder (eher erforderlich) verkleinert werden kann:

```
font-size: 0.95em;  /* Basisschriftgröße nun ca. 15px */
```

6.1.2 Die Überschriften

Überschriften betiteln längere Texte nicht nur, sondern gliedern sie auch. Die zwingende Zuordnung eines Textabsatzes zu einer Hierarchiestufe (im Sinne einer Dokumentgliederung in Abschnitte) ist in HTML zwar nicht modellierbar, jedoch sollte man Überschriften grundsätzlich in folgendem Sinne einsetzen:

- Die Überschrift erster Ordnung betitelt den Dokumentinhalt
- Überschriften zweiter, dritter und weiterer Ordnungen betiteln der Reihe nach nachfolgende, niederrangigere Dokumentabschnitte
- Bei der Betitelung soll keine Überschriftenstufe übersprungen werden

Es ergibt sich eine Hierarchiefolge, wie sie in Abbildung 6.1 angedeutet ist. Achtung – die Umrandungen bezeichnen die Hierarchiestufe, nicht den Bereich des Überschriftencontainers! Beachten Sie, dass daher **keine wirkliche Zuordnung** der `<p>`-Container zu einer voranstehenden Überschrift existiert.

Erst XHTML 2.0 verspricht dieses Problem durch Einführung eines `<section>`-Containers zu lösen[2], der Überschriften und Absätze einer Stufe zusammenfasst (Sie könnten dies natürlich durch `<div>`-Container oder auf vergleichbarem Wege auch in HTML halbwegs nachbauen – früher wurde zu diesem Zweck auch die horizontale Trennline `<hr>` eingesetzt).

Die Beachtung der Überschriftenhierachie ist eine Forderung der **Barrierefreiheit** (laut WAI 3.5, BITV 3.5), da Überschriften von Screenreadern und vergleichbaren Hilfsmitteln auch zur Kenntlichmachung der Dokumentgliederung und als Navigationshilfe einsetzen.

Ebenfalls abzulehnen ist der Einsatz von Überschriften aus Präsentationsgründen (Textvergrößerung). Analog sollte nicht vergrößerter Text an Stelle einer Überschrift eingesetzt werden. CSS bietet hinreichend Mittel, um Schriftgrößen gegebenenfalls anzupassen.

2 Siehe: http://www.w3.org/TR/xhtml2/mod-structural.html#edef_structural_section.

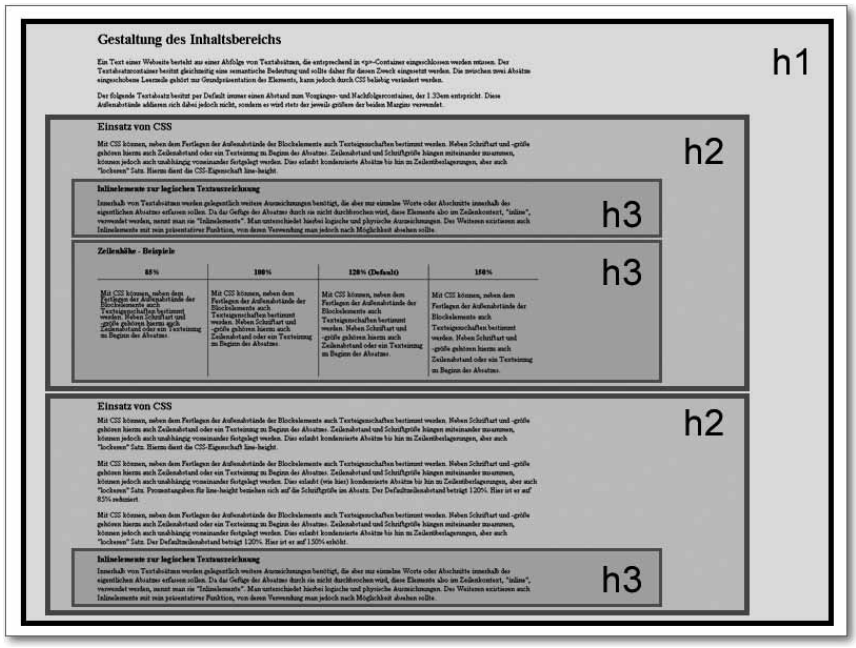

Abbildung 6.1 Überschriftenhierarchie in einem HTML-Dokument

Praxistipp – Überschriften als Linkziel

Wenn Sie den Überschriften Ihres Dokuments konsequent `id`-Attribute zuweisen, können Sie auf einfachen Wege **Sprunglinks** zu einzelnen Dokumentabschnitten erstellen.

6.1.3 Der Textabsatzcontainer

Ein Text einer Webseite besteht aus einer Abfolge von Textabsätzen, die entsprechend in `<p>`-Container eingeschlossen werden müssen. Der Textabsatzcontainer besitzt gleichzeitig eine **semantische Bedeutung** und sollte daher nur für diesen Zweck eingesetzt werden.

Die zwischen zwei Absätze eingeschobene Leerzeile gehört zur Grundpräsentation des Elements, die jedoch durch CSS beliebig verändert werden kann. Der folgende Textabsatz besitzt per Default immer einen **Abstand** zum Vorgänger- und Nachfolgercontainer, der 1,12em entspricht, also auf der aktuell für den Container gültigen Schriftgröße basiert.

Die offiziellen Defaultmargins für den `<p>`-Container lauten wie folgt:

```
p  { margin: 1.12em 0; }
```

… was sich als »Außenabstand von 1,12em nach oben und unten sowie Außenabstand 0 nach rechts und links« übersetzen lässt. Sie können selbstverständlich beliebige andere Werte vergeben, sofern Ihr Layout dies erfordert.

Die **Außenabstände** nach oben und unten zweier aufeinanderfolgender Container addieren sich dabei jedoch nicht, sondern es wird stets der jeweils größere der beiden Margins verwendet (dies gilt beispielsweise, wenn ein Textabsatz auf eine Überschrift folgt, die einen größeren Bottommargin-Wert besitzt).

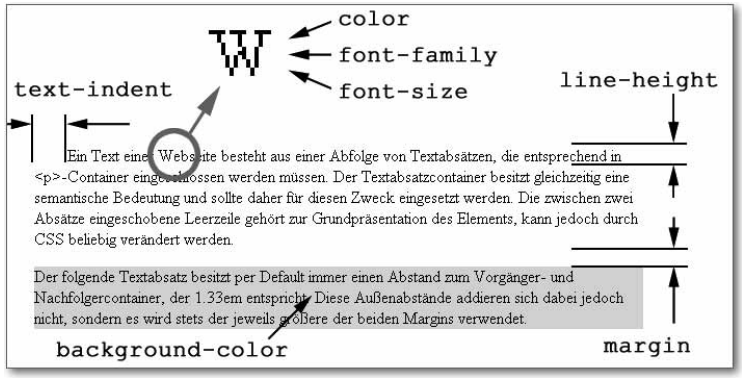

Abbildung 6.2 Die wichtigsten CSS-Parameter für Textabsätze

Mit CSS können, neben dem Festlegen der Außenabstände der Blockelemente auch Texteigenschaften bestimmt werden. Neben Schriftart und -größe gehören hierzu auch **Zeilenabstand** oder ein **Texteinzug** zu Beginn des Absatzes. Zeilenabstand und Schriftgröße hängen miteinander zusammen, können jedoch auch unabhängig voneinander festgelegt werden. Dies erlaubt eng gesetzte Absätze (»kompress«) bis hin zu Zeilenüberlagerungen, aber auch »lockeren« Satz. Hierzu dient die CSS-Eigenschaft `line-height`.

Angaben für `line-height` beziehen sich auf die Schriftgröße innerhalb des Absatzes. Die Defaultzeilenhöhe beträgt 120 % der Schriftgröße. Eine Prozentangabe unter 120 % reduziert daher die Zeilenhöhe, eine Angabe darüber vergrößert sie. Grundsätzlich sollte man mit einer Reduzierung der Zeilenhöhe vorsichtig umgehen, da dies schnell die Lesbarkeit des Textes erschwert (der Wert 100 % sollte nicht unterschritten werden). Eine Erhöhung ist im Allgemeinen unkritischer, sofern die Zeilenlänge nicht zu kurz ist, da der Textblock oberhalb von 150 % sonst optisch zu zerfallen droht (siehe Abbildung 6.3).

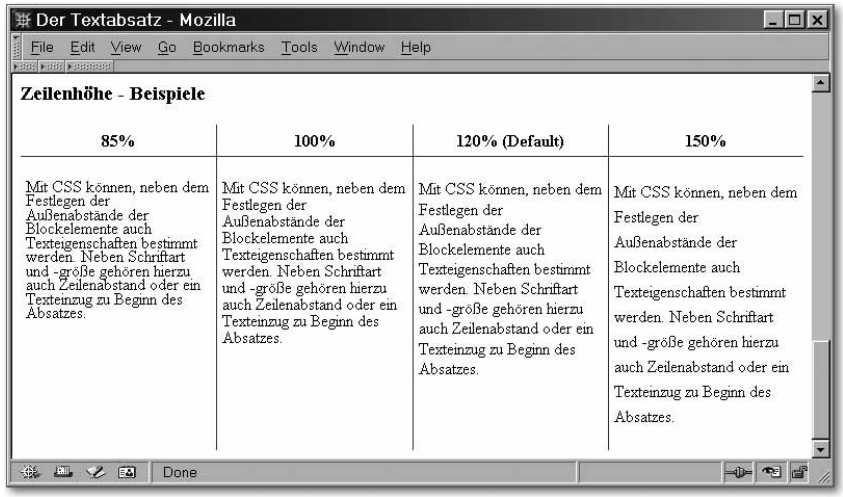

Abbildung 6.3 Verschiedene Zeilenhöhen mit line-height im Vergleich

6.2 Inlineelemente zur Textauszeichnung

Innerhalb von Textabsätzen werden gelegentlich weitere Auszeichnungen benötigt, die aber nur einzelne Worte oder Abschnitte innerhalb des eigentlichen Absatzes erfassen sollen. Da das Gefüge des Absatzes durch sie nicht durchbrochen wird, diese Elemente also im Zeilenkontext, »inline«, verwendet werden, nennt man sie **Inlineelemente**.

Man unterschiedet hierbei **logische** und **physische** Auszeichnungen, welche Rolle oder Bedeutung ihrer Inhalte kennzeichnen, ohne damit notwendigerweise eine Präsentation festzulegen. Des Weiteren existieren auch Inlineelemente mit rein präsentativer Funktion, von deren Verwendung man jedoch nach Möglichkeit zugunsten von CSS absehen sollte.

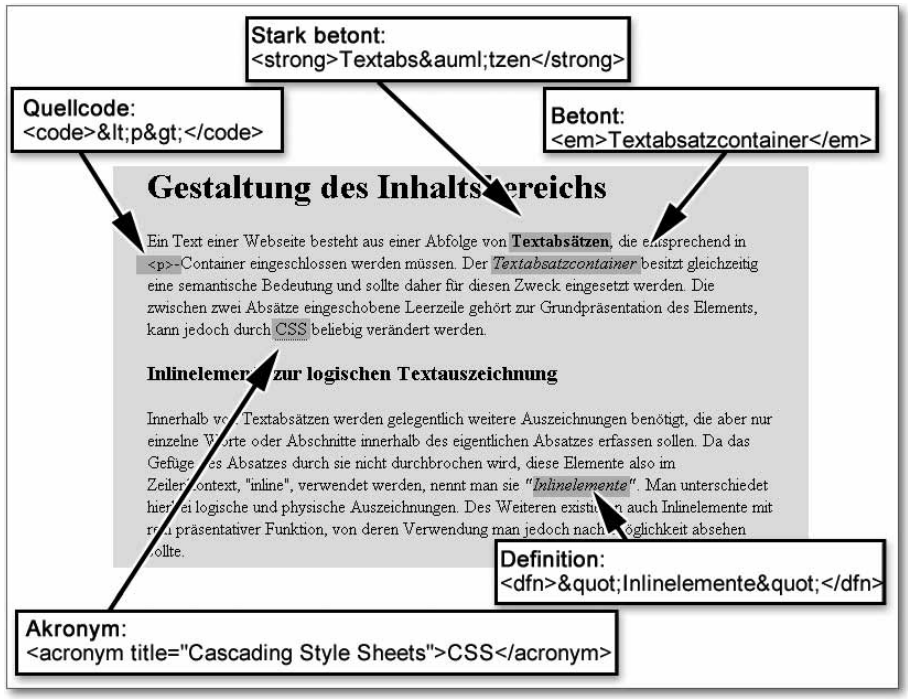

Abbildung 6.4 Verschiedene logische Textauszeichnungen

Logische und physische Auszeichnung

In der folgenden Tabelle sind Inlineelemente zur logischen und physischen Textauszeichnung aufgelistet. Ausführliche Erläuterungen zu den einzelnen Elementen finden Sie in Kapitel 14, *(X)HTML-Elementreferenz*, in den angegebenen Abschnitten – an dieser Stelle möchte ich mich auf grundlegendere Aspekte beschränken.

Element	Rolle	Erläuterung
`<cite> ... </cite>`	Bezeichnet die Quelle (Herkunft) eines Zitats	Abschnitt 14.18
`<code> ... </code>`	Bezeichnet Quellcode; gewöhnlich mit nicht proportionaler Schrift dargestellt	Abschnitt 14.19
`<dfn> ... </dfn>`	Bezeichnet eine Definition; gewöhnlich kursiv dargestellt	Abschnitt 14.24
`<em> ... </em>`	Einfache Hervorhebung; gewöhnlich kursiv dargestellt	Abschnitt 14.29
`<kbd> ... </kbd>`	Bezeichnet eine Tastatureingabe;	Abschnitt 14.46

Tabelle 6.2 Inlineelemente zur logischen und physischen Textauszeichnung

Element	Rolle	Erläuterung
`<q> ... </q>`	Bezeichnet ein Inlinezitat durch Anführungszeichen	Abschnitt 14.67
`<samp> ... </samp>`	"sample"; Auszug z. B. eines Programmquellcodes	Abschnitt 14.69
`<strong> ... </strong>`	Bezeichnet starke Hervorhebung; gewöhnlich fett dargestellt	Abschnitt 14.75
`<var> ... </var>`	Bezeichnet eine Variable	Abschnitt 14.91
Attribute (alle)	`class`, `id`, `style`, `title` (Universalattribute)	

Tabelle 6.2 Inlineelemente zur logischen und physischen Textauszeichnung (Forts.)

Die hier vorgestellten Inlineelemente haben gemeinsam, dass sie nicht auf eine spezielle Präsentation ihrer Inhalte abzielen (obwohl diese teilweise durchaus damit verbunden ist), sondern eine Aussage über ihre Bedeutung machen. Dies macht ihren Einsatz prinzipiell **unabhängig vom auswertenden User Agent** – ist dieser beispielsweise ein Screenreader, so kann er beispielsweise eine Betonung mit `<strong>` auswerten (durch eine Erhöhung der Leselautstärke), wohingegen eine einfache Fettauszeichnung mit `<b>` keine zwingende sematische Bedeutung besitzt.

Element	Rolle	Erläuterung
`<span> ... </span>`	Neutraler Container; wichtig in Zusammenhang mit Stylesheets	Abschnitt 14.73
Attribute	`class`, `id`, `style`, `title` (Universalattribute)	

Tabelle 6.3 Der -Container

Eine Sonderstellung in diesem Kontext bildet der neutrale `<span>`-Container, der als Hauptaufgabe das Tragen einer CSS-Klasse übernimmt, also eine Möglichkeit darstellt, eine reine Styleinformation ohne Zweckentfremdung eines anderen Elemente an einen Inlinebereich zu binden. Aus diesem Grunde könnte man seine Wirkung als »präsentativ« bezeichnen, jedoch kann das Element über entsprechende Klassennamen auch logische Bedeutungen vermitteln:

```
<p>Bauteil <span class="bestellnummer">08-15</span>:
<span class="preis">22.95</span></p>
```

Auch die **Barrierefreiheit** eines Textes lässt sich durch Auszeichnung mit logischen Elementen verbessern. Hier sind in erster Linie die beiden Elemente `<abbr>` und `<acronym>` zu nennen, die neben der Kennzeichnung auch gleich die Langfassung eines Begriffes zur Verfügung stellen (über ihr `title`-Attribut). Ein Screenreader kann so eine »unaussprechliche« Abkürzung durch Klartext erset-

zen, ein normaler Browser markiert die Begriffe lediglich als »mit Erläuterung hinterlegt« (dies geschieht durch Unterpunktion; die Erläuterung erscheint bei Mouseover als Tooltip – beachten Sie jedoch, dass der Internet Explorer das Element `<abbr>` erst ab Version 7 unterstützt).

Element	Rolle	Erläuterung
`<abbr> ... </abbr>`	Auszeichnung als Abkürzung	Abschnitt 14.2
`<acronym> ... </acronym>`	Auszeichnung als Akronym	Abschnitt 14.3
Attribute (alle)	`class`, `id`, `style`, `title` (Universalattribute)	

Tabelle 6.4 Logische Inlineelemente zur Unterstützung der Barrierefreiheit

Zwei weitere logische Elemente dienen der Anbringung von **Änderungsmarken** im Text, denen auch Urheberangaben und Datum hinzugefügt werden können. Es existiert zwar eine Defaultpräsentation, die als gelöscht gekennzeichnete Passagen durchgestrichen und eingefügte mit Unterstrich darstellt, jedoch kann mit CSS auch eine beliebig andere Darstellung erzwungen werden. Folgende CSS-Befehle blenden gestrichene Passagen aus und highlighten Einfügungen durch gelben Hintergrund (der Unterstrich wird entfernt):

```
del  { display:none; }
ins  { text-decoration:none; background-color: yellow; }
```

Element	Rolle	Erläuterung
`<del> ... </del>`	"deleted"; markiert einen Textabschnitt als gestrichen	Abschnitt 14.23
`<ins> ... </ins>`	"inserted"; markiert einen Textabschnitt als hinzugefügt	Abschnitt 14.44
Attribute (alle)	`cite`, `datetime`	

Tabelle 6.5 Logische Inlineelemente für Änderungsmarkierungen

Präsentative Textauszeichnung

Weniger gern gesehen ist heutzutage eine Auszeichnung, die auf eine **reine Präsentationswirkung** ausgerichtet ist – man kann mit ihnen zwar in normalen Browsern eine zu logischer Auszeichnung identische optische Wirkung erzielen, auf der Strecke bleibt jedoch die semantische Bedeutung der Inhalte.

Beispielsweise ist die Umsetzung von `<strong>Textabsätzen</strong>` und `<b>Textabsätzen</b>` im Browser identisch, ebenso die von `<code><p></code>` und `<tt><p></tt>`. Wo `<strong>` eine Wichtigkeit im Sinne einer Betonung festlegt, markiert `<b>` lapidar eine (prinzipiell aus-

sagefreie) Fettauszeichnung. Das Element `<code>` weist seinem Inhalt die Bedeutung »Quelltextausschnitt« zu, wohingegen `<tt>` lediglich nicht-proportionale Schrift fordert – ohne Begründung hierfür.

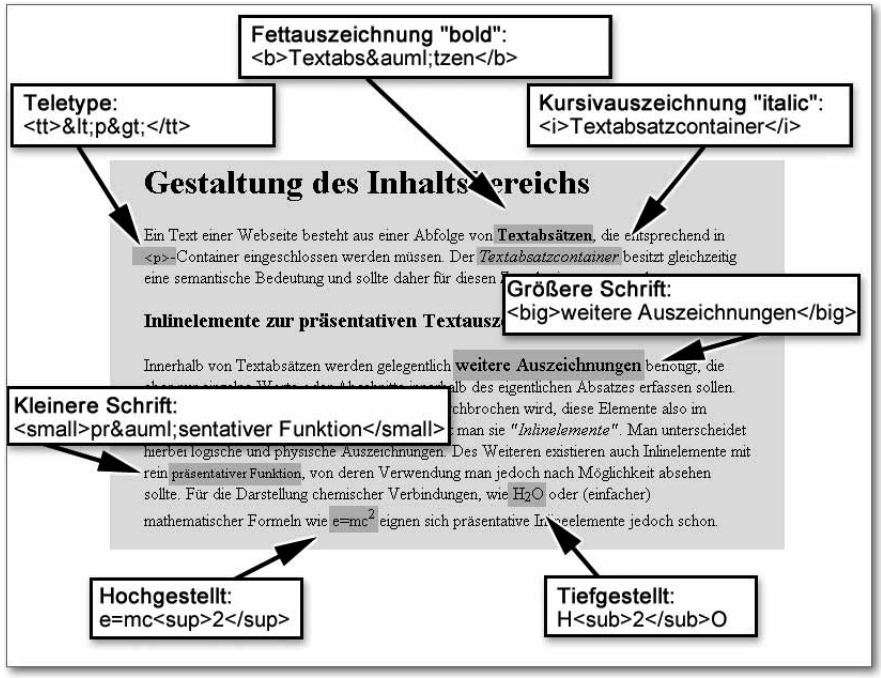

Abbildung 6.5 Verschiedene präsentative Inlineauszeichnungen

Generell ist gegen präsentative Auszeichnung nichts einzuwenden, solange damit keine Aussage über die Bedeutung der Inhalte verbunden werden soll: User Agents, die nicht visuell orientiert sind, werden diese Elemente nicht sinnvoll auswerten können und gegebenenfalls ignorieren. Screenreader können das *Vorhandensein* von Präsentationsmarkup zwar ansagen, müssen dessen Interpretation (»*Warum* ist dieser Abschnitt fett?«) jedoch dem Nutzer überlassen.

Element	Rolle	Erläuterung
`<b> ... </b>`	"bold"; Fettauszeichnung;	Abschnitt 14.7
`<big> ... </big>`	erhöht die Schriftgröße um eine Stufe (auf ca. 120 %)	Abschnitt 14.11
`<i> ... </i>`	"italic"; Kursivauszeichnung	Abschnitt 14.40
`<small> ... </small>`	reduziert die Schriftgröße um eine Stufe (auf ca. 80 %)	Abschnitt 14.72

Tabelle 6.6 Textauszeichnungen mit vorwiegend präsentativem Character

Element	Rolle	Erläuterung
`<sub> ... </sub>`	"subscript"; Inhalt wird tiefgestellt und kleiner präsentiert	Abschnitt 14.77
`<sup> ... </sup>`	"superscript"; Inhalt wird hochgestellt und kleiner präsentiert	Abschnitt 14.78
`<tt> ... </tt>`	"teletype"; Inhalt wird mit nicht proportionaler Schrift dargestellt	Abschnitt 14.88

Tabelle 6.6 Textauszeichnungen mit vorwiegend präsentativem Character (Forts.)

Es existieren noch weitere Elemente, die entweder aus den Anfangstagen von HTML stammen und sich überlebt haben, oder missverständlich sind. Ein Paradebeispiel ist der `<font>`-Container, dessen Aufgabe der Schriftformatierung gänzlich durch CSS übernommen werden können (gegebenenfalls durch einen `<span>`-Container in Verbindung mit einer CSS-Klasse). Zu nennen sind auch `<s>` und `<strike>` (hier nicht aufgeführt), an deren Stelle besser die Änderungsmarke `<del>` eingesetzt werden sollte, oder die Unterstreichungsmarke `<u>`, die zur Verwechselung des Inhaltes mit einem Hyperlink führen kann und deren Wirkung, wenn nötig, ebenfalls durch CSS nachgebildet werden sollte.

Element	Rolle	Erläuterung
`<font> ... </font>`	Präsentiert den eingeschlossenen Textinhalt durch Angabe von Schriftgröße, -typ und -farbe	Abschnitt 14.32
`<s> ... </s>`	"strikethrough";präsentiert den eingeschlossenen Inhalt durchgestrichen	Abschnitt 14.68
`<u> ... </u>`	"underline"; präsentiert den eingeschlossenen Textinhalt mit Unterstrich	Abschnitt 14.89

Tabelle 6.7 Präsentationselemente, die nicht länger eingesetzt werden sollten

6.3 Der Hyperlinkcontainer, revisited

Bereits angerissen wurde der **Hyperlinkcontainer** `<a>` der zur Erstellung von Verknüpfungen zwischen Dokumenten dient. Der »Ankercontainer«, wie er auch genannt wird, zählt ebenfalls zu den Inlineelementen. Der Begriff »Anker« stammt dabei aus der Hypertexttheorie, wo mit »Ankerpunkt« das Textfragment bezeichnet wird, an welchem die eigentliche Verknüpfung »verankert« wird. Ein Hypertext-Anker besitzt demnach zwei Aspekte[3]:

3 Zur Theorie siehe: François Fluckiger, *Multimedia im Netz*; Prentice Hall 1996.

1. **Als Ankerpunkt:**
   »Ein verknüpfbares Informationsfragment.« (z. B. der Linktext)
2. **Als Verknüpfung:**
   »Ein Bezug zu einer Information.« (das Linkziel, z. B. als URI)

In HTML entspricht der Ankerpunkt dem in den Container eingeschlossenen Text, das Verknüpfungsziel wird als Wert des `href`-Attributs definiert:

```
<a href="zieldatei.html">Dies ist der Ankertext</a>
```

Das andere, mit `<a>` früher oft eingesetzte Attribut `name` hat an Bedeutung eingebüßt – es diente dazu, einen Ankercontainer auch zum Ziel eines Links machen zu können. Diese Funktion hat mittlerweile das `id`-Attribut übernommen, mit dem Vorteil, dass nun außer `<a>`-Containern beliebige Elementcontainer als Linkziel fungieren können.

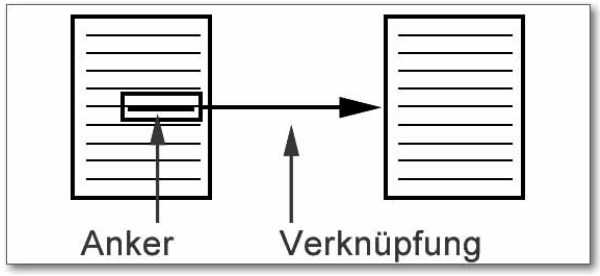

Abbildung 6.6 Der Textinhalt von <a> dient als so genannter »Anker«.

Element	Rolle	Erläuterung
`<a> ... </a>`	"anchor"; bezeichnet Quelle oder Ziel eines Hyperlinks	Abschnitt 14.1
Attribute (Auswahl)	`accesskey, href, name, tabindex, target, title`	

Tabelle 6.8 Der <a>-Container zum Erzeugen von Hyperlinks

Die Defaultpräsentation des Textinhalts eines Hyperlinks ist unterstrichen. Innerhalb von Fließtexten sollte dies beibehalten werden – das Styling von Navigationslinks innerhalb von Menüs ist eine andere Sache und wird im folgenden Kapitel vertieft. Kennzeichnen Sie die Links innerhalb des Inhaltsbereichs auf eine Weise, dass sie erkennbar bleiben. Sie können Styleklassen einsetzen, um verschiedene Rollen der Links zu verdeutlichen, wie siteintern, extern oder E-Mail-Link (siehe Abbildung 6.7).

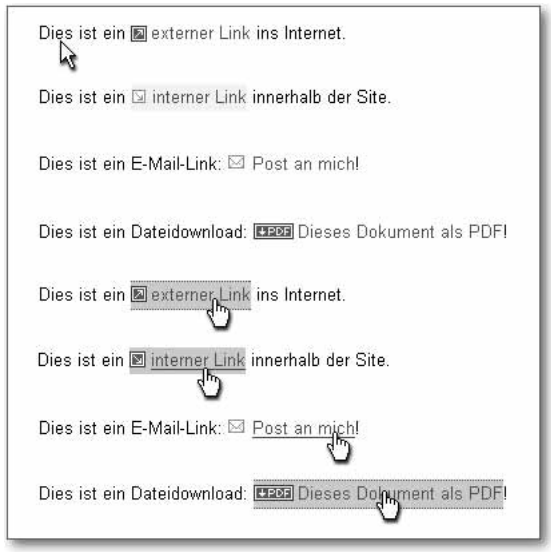

Abbildung 6.7 Hyperlinks mit Icons, unten Hover-Zustände

Den Hyperlinks in Abbildung 6.7 liegt folgender Quellcode zugrunde:

```
<p>Dies ist ein <a href="http:www.weitweg.de"
 class="extern">externer Link</a> ins Internet.</p>

<p>Dies ist ein <a href="andere_seite.html"
 class="intern">interner Link</a> innerhalb der Site.</p>

<p>Dies ist ein E-Mail-Link: <a href="mailto:x@y.de"
 class="mail">Post an mich</a>!</p>

<p>Dies ist ein Dateidownload: <a href="dokument.pdf"
 class="pdf">Dieses Dokument als PDF</a>!</p>
```

Listing 6.1 Verschiedene Hyperlinks mit Klassenattribut

Ohne CSS würden die Links präsentiert, wie in Abbildung 6.8; hier mehrere exemplarische Auszüge der verwendeten CSS-Anweisungen. Die nicht vorgestellten Klassen arbeiten analog, jedoch mit anderen Icon-Grafiken. Entsprechend müssen die `padding`-Werte angepasst werden, um die Breiten zu kompensieren.

```
a.intern {
  text-decoration:none;
  background:#f5f5f5 url('intern.gif')0em 0.2em no-repeat;
  padding:1px 0 3px 16px; color:#D13500;
}
```

```
a.intern:hover {
 text-decoration:underline;
 background:#ffccbb url('intern_a.gif') 0em 0.2em no-repeat;
}

a.mail {
  text-decoration:none;
  background:url('mail_link.gif') 0em 0.25em no-repeat;
  padding:1px 0 3px 18px; color:#D13500;
}

a.mail:hover { text-decoration:underline;
}
```

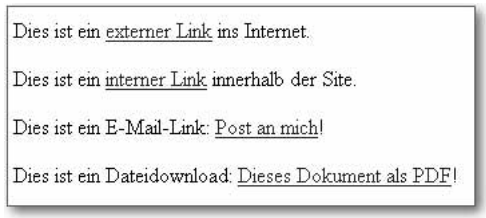

Abbildung 6.8 Die Hyperlinks von Abbildung 6.7 ohne CSS-Präsentation

6.3.1 Welche Stylingmöglichkeiten bietet CSS für Hyperlinks?

Speziell in Zusammenhang mit Hyperlinks kann das `text-decoration`-Property eingesetzt werden, um den oft ungeliebten **Unterstrich** des Links zu entfernen. Innerhalb von Fließtexten ist davon abzuraten, sofern nicht eine andere Methode (wie eine Hintergrundfarbe) eingesetzt wird, den Link optisch zu kennzeichnen. Mittels des `background-image`-Properties kann ein Link ein **Icon** in Form eines Hintergrundbildes erhalten. Schaffen Sie mit dem `padding`-Property genügend Platz, damit der Linktext das Icon nicht verdeckt.

Die interessanteste Stylingmöglichkeit per CSS bietet der Pseudoselektor `:hover`, der Mouseover-Effekte für Links erzeugen kann. Für den Hoverzustand des Links können Sie sämtliche Darstellungsparameter verändern, also Unterstrich einblenden, Farbänderung, anderes Hintergrundbild. Sie erzielen so ohne Einsatz von JavaScript dynamische Linkpräsentationen.

6.3.2 Welche HTML-Attribute werden für Hyperlinks benötigt?

Unabdingbar ist das `href`-Attribut, das die verknüpfte Ressource nennt. Einem barrierefreien Link werden Sie über das `title`-Attribut weitere Information über das Linkziel mit auf den Weg geben. Zusätzlich können Sie mit `accesskey` oder

tabindex die Zugänglichkeit über Tastatur vereinfachen. Im Rahmen von Framesets oder zum einfachen Öffnen neuer Fenster (Vorsicht, dies ist nur bedingt barrierefrei), können Sie das target-Attribut einsetzen. Es steht allerdings nur in den *transitional*- und *frameset*-Versionen von HTML und XHTML zur Verfügung. In XHTML 1.1. existiert es nicht.

6.4 Eingebettete Medien

Da HTML ein reines Textformat ist, können Nicht-Text-Daten nicht unmittelbar Teil des Dokuments sein. Stattdessen muss ein **Verweis** auf diese, dann extern liegende Datei vorgenommen werden, die den Browser veranlasst, die Datei an der gewünschten Stelle in das Dokumentlayout aufzunehmen. Medieneinbindungen sind also Platzhalter.

Das wichtigste dieser Elemente ist der leere <img/>-Tag, der eine Grafik einfügt – meist ist diese vom Typ JPG, GIF oder PNG. Die **einzubindende Datei** wird über das src-Attribut des Elements bestimmt, das obligatorische alt-Attribut stellt einen **Alternativtext** zur Verfügung, wenn die Grafik nicht angezeigt werden kann. Über die Attribute height und width werden die physischen Abmessungen der Grafik in Pixel angegeben.

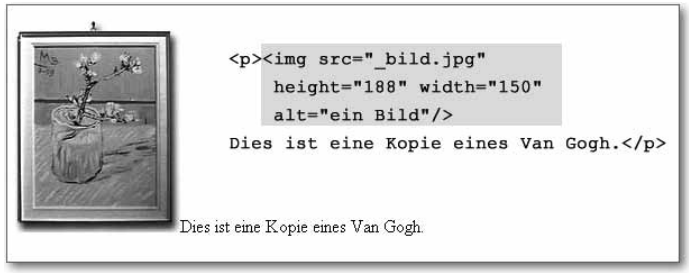

Abbildung 6.9 Einfügen einer Grafik

Das Bild erscheint als Inlineelement im Zeilenkontext, bestimmt daher also die Zeilenhöhe. Mittels des Float-Properties kann es von längeren Textpassagen umflossen werden. Da das Bild nicht mehr im Zeilenkontext steht, beeinflusst es die Zeilenhöhe nicht länger.

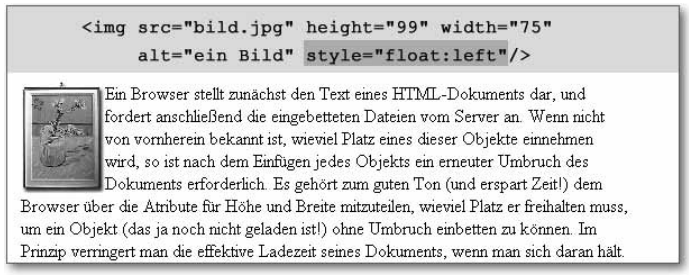

Abbildung 6.10 Ein gefloatetes Bild in einem Text.

Element	Rolle	Erläuterung
`<img />`	Bindet eine Grafikdatei ein	Abschnitt 14.42
Attribute (Auswahl)	**`alt`**, `border`, `height`, `longdesc`, `name`, **`src`**, `width`	

Tabelle 6.9 Das <img>-Element zum Einbetten von Grafiken

Als weitere Elemente zum Einbetten von Medien dienen die Container `<object>` und `<iframe>`, wobei `<object>` beliebige Medienobjekte, also Audio, Video und weitere Datenformate (auch Text oder Grafiken) einbetten kann. Über `<param />`-Elemente können Werte, wie Abspielparameter an ein Medienobjekt weitergegeben werden.

Sonderfälle

Element	Rolle	Erläuterung
`<object> ... </object>`	Fügt ein Objekt in die Seite ein	Abschnitt 14.59
Attribute (Auswahl)	`archive`, `classid`, `codebase`, `data`, `height`, `width`	
`<param />`	Übergibt Parameter an den umgebenden `<object>`-Container	Abschnitt 14.64
Attribute (Auswahl)	**`name`**, `value`	

Tabelle 6.10 Das <object>-Element zum Einbetten beliebiger Medien

Inlineframes dienen zum Einfügen von HTML-Dokumenten in eine Webseite. Dies geschieht durch Einbetten in einen Framerahmen, dessen Position durch den `<iframe>`-Container bestimmt wird. Die Verwendung von IFrames gilt aus Gründen der **Barrierefreiheit** als problematisch.

Element	Rolle	Erläuterung
`<iframe> ... </iframe>`	Definiert einen Inline Frame	Abschnitt 14.41
Attribute (Auswahl)	`frameborder`, `height`, `name`, `scrolling`, `src`, `width`	

Tabelle 6.11 Das <iframe>-Element zum Einbetten von HTML-Dokumenten

6.4.1 Welche Stylingmöglichkeiten bietet CSS für eingebettete Medien?

Generell können Sie die **Box**- und **Floatproperties** einsetzen, wobei Rahmen und Außenabstände hier am wichtigsten sind. Hintergrundfarben können bei teiltransparenten Grafiken von Interesse sein, oder zum Styling des Fallbackinhalts eines `<object>`-Containers. Normalerweise wird eine Farbdefinition jedoch durch das Medienobjekt verdeckt werden.

6.4.2 Welche HTML-Attribute werden für eingebettete Medien benötigt?

Neben den spezifischen Attributen, die das Medium einbetten (wie das `src`-Attribut bei `<img/>`) sollten Sie die **physischen Abmessungen** der Objekte per HTML-Attribut angeben (`width` und `height`), da diese Informationen unabhängig von CSS vorliegen sollten, um den Renderingvorgang zu beschleunigen.

Des Weiteren wichtig und **teilweise vorgeschrieben** sind Attribute, die die Barrierefreiheit der Inhalte betreffen – hier ist in erster Linie das `alt`-Attribut von `<img/>` zu nennen, dass einen Alternativtext bei fehlender oder deaktivierter Grafik angibt. Das `border`-Attribut für `<img/>` benötigen Sie eigentlich nicht, es kann jedoch von Interesse sein, den Rahmen über `border="0"` für solche Grafiken zu deaktivieren, die sich innerhalb von Hyperlink-Containern befinden. Zwar zählt `border` zu den Präsentationsanweisungen, die auch per CSS verfügbar sind, steht so aber auch ohne Stylesheet zur Verfügung.

6.5 Listen

Listen stellen in HTML eine eigene Gattung von Blockelementen dar, die, Textabsätzen gleichgestellt, innerhalb des Dokumentinhalts auftreten. In diesem Falle dienen sie dem Erzeugen von Aufzählungen, deren Listenelemente `<li>` wahlweise nummeriert (innerhalb von »geordneten Listen«, `<ol>`) oder mit Bullets versehen werden (innerhalb von »ungeordneten Listen«, `<ul>`).

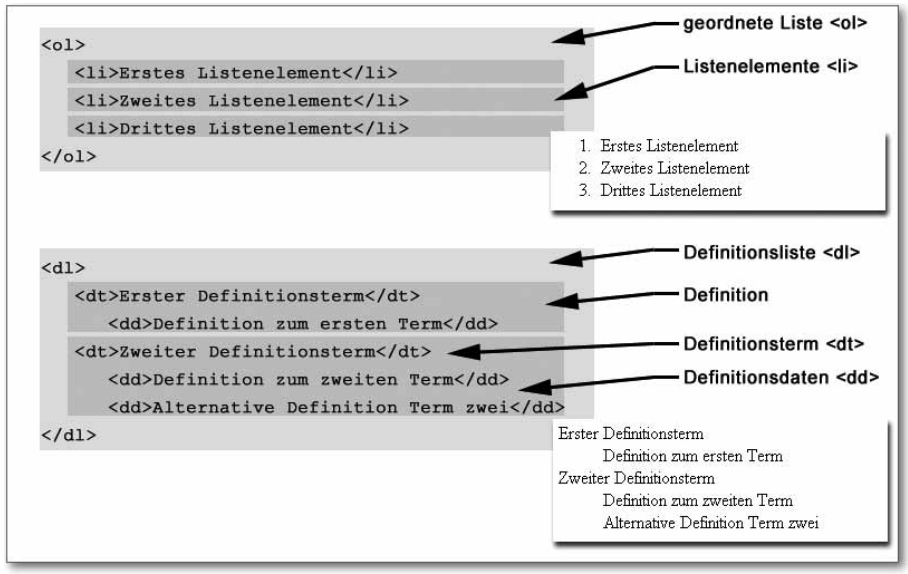

Abbildung 6.11 Geordnete Liste, Definitionsliste

Als dritter, noch gebräuchlicher Listentyp existiert die so genannte Definitionsliste `<dl>`, die allerdings keine Listenelemente vom Typ `<li>`, sondern abwechselnd (aber ohne festgeschriebene Anzahl) definierte Begriffe (»definition term«, `<dt>`) und deren Definition (»definition data«, `<dd>`) enthält.

Definitionslisten

Für **Definitionslisten** existieren daher drei Elemente. Insgesamt wird dieser Listentyp eher selten verwendet, bietet jedoch interessante Einsatzmöglichkeiten, beispielsweise in Zusammenhang mit dem Gestalten von Formularen.

Element	Rolle	Erläuterung
`<dl> ... </dl>`	"definition list"; Container für eine Definitionsliste; enthält `<dt>` und `<dl>`-Container	Abschnitt 14.27
`<dt> ... </dt>`	"definition term"; bezeichnet den zu definierenden Ausdruck einer Definition	Abschnitt 14.28
`<dd> ... </dd>`	"definition data"; bezeichnet die zum vorstehenden `<dt>`-Container gehörende Definition	Abschnitt 14.22

Tabelle 6.12 Elemente für Definitionslisten

Geordnete und ungeordnete Listen

Die meistverbreiteten Listen sind die geordnete und die ungeordnete Liste, die sich nur durch das umgebende Containerelement unterscheiden. **Geordnete Listen** werden benötigt, wo neben einer Hierarchie auch eine **Nummerierung** der Listenpunkte gefordert ist. **Ungeordnete Listen** zeigen keine Nummerierung, sondern stellen die Listenelemente mit **Bullets** dar.

Element	Rolle	Erläuterung
`<ul> ... </ul>`	"unordered list"; Container für eine ungeordnete Liste; Listenelemente `<li>` werden durch Bullets gekennzeichnet	Abschnitt 14.90
`<ol> ... </ol>`	"ordered list"; Container für eine geordnete Liste; Listenelemente `<li>` erhalten Nummerierung mit Ziffern oder Buchstaben	Abschnitt 14.69
`<li> ... </li>`	"list item"; Listenelement einer ungeordneten oder geordneten Liste; Darstellung abhängig von der Art des umgebenden Listencontaners	Abschnitt 14.49

Tabelle 6.13 Elemente für geordnete und ungeordnete Listen

Hierarchien werden bei beiden Listentypen durch **Verschachtelung** erzeugt, wobei die Typen auch gemischt werden dürfen. Achten Sie bei der Verschachtelung darauf, dass die jeweils innenliegende Liste komplett in einem `<li>`-Container der jeweils äußeren Liste enthalten sein muss (siehe Abbildung 6.12).

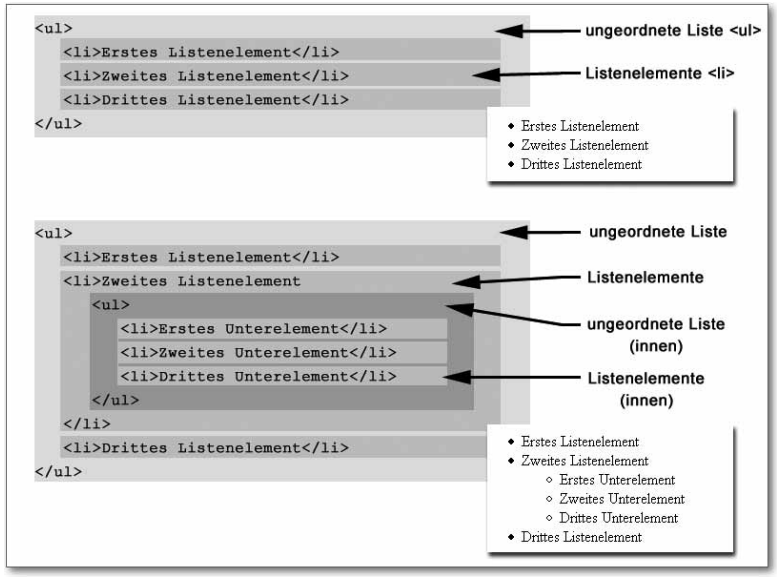

Abbildung 6.12 Ungeordnete Listen; einfach und verschachtelt

6.5.1 Welche Stylingmöglichkeiten bietet CSS für Listen?

CSS bietet speziell für Listen das Sammelproperty `list-style` und dazugehörige Unterproperties, um das Erscheinungsbild von `<ol>`- und `<ul>`-Listen anzupassen. Sie können die Aufzählungselemente deaktivieren (`list-style-type: none`) oder beliebig andere, bis hin zu grafischen Elementen, definieren.

Ansonsten gelten für Listen alle **Boxproperties**, mit denen Sie Abmessungen, Innen- und Außenabstände, Rahmen und Hintergrundfarben definieren können. In Zusammenhang mit Navigationslisten (mehr dazu im folgenden Kapitel) spielen auch das **Display-Property** und **Float-Eigenschaften** eine Rolle.

6.5.2 Welche HTML-Attribute werden für Listen benötigt?

Die kurze Antwort: keine. Es ist zwar für `<ul>`, `<ol>` und `<li>` ein Attribut `type` definiert, um die Art des Aufzählungselements festzulegen, jedoch ist das CSS-Property `list-style-type` diesem überlegen. Allenfalls mag das Attribut `start` für `<ol>` erwägenswert sein, falls eine Nummerierung nicht bei 1 sondern mit einer anderen Ziffer beginnen soll.

6.6 Tabellen

Tabellen wurden in der Vergangenheit häufig nicht im Sinne ihrer eigentlichen Bestimmung eingesetzt, sondern, aufgrund der seinerzeit dürftigen Layout-Fähigkeiten von HTML fast immer zum Seitenlayout. Seit reine CSS-Layouts durch breite Browserunterstützung möglich sind, ist die Tabelle von Layoutaufgaben entbunden und kann wieder zu dem Zweck eingesetzt werden, für den sie eigentlich konzipiert wurde: zur Tabellierung von Daten (Abbildung 6.13).

Abgasgrenzwerte für Pkw

Treibstoff	Schadstoff	EWG Stufe 1 (seit 1992)	EG Stufe 2 (seit 1996)	Euro 3 (seit 2000)	Euro 4 (seit 2005)	Euro 5 (Vorschlag)
Benzin	Kohlenmonoxid (CO)	2 720	2 200	2 300	1 000	1 000
	Kohlenwasserstoffe (HC)	-	-	200	100	75
	Stickoxid (NO_x)	-	-	150	80	60
	HC + NO_x	970	500	-	-	-
Diesel	Kohlenmonoxid (CO)	2 720	1 000	640	500	500
	Stickoxid (NO_x)	-	-	500	250	200
	HC + NO_x	-	700/900 DI	560	300	250
	Partikel	140	80/100 DI	50	25	5

Abbildung 6.13 Eine komplexere, mit CSS gestylte Datentabelle

Der Grundaufbau einer Tabelle ist verhältnismäßig einfach – man benötigt lediglich drei Arten von Elementen: den **Tabellencontainer** `<table>`, die **Zeilencontainer** `<tr>` und die **Datenzellencontainer** `<td>`. Sofern man Zellen in Tabellenkopffunktion benötigt, kommt ein viertes Element `<th>` hinzu, das ebenfalls eine Datenzelle darstellt, jedoch als **Headerzelle**....

Element	Rolle	Erläuterung
`<table>` ... `</table>`	Definiert eine Tabelle	Abschnitt 14.79
Attribute (Auswahl)	`cellpadding`, `cellspacing`, `frame`, `rules`, `summary`	
`<tr>` ... `</tr>`	Definiert eine Tabellenzeile als Unterelement von `<table>`	Abschnitt 14.87
Attribute (Auswahl)	`align`, `valign`	
`<th> ... </th>`	Definiert eine Tabellenkopfzelle als Unterelement von `<tr>`	Abschnitt 14.84
Attribute (Auswahl)	`abbr`, `align`, `axis`, `colspan`, `height`, `rowspan`, `scope`, `valign`	
`<td> ... </td>`	Definiert eine Tabellenzelle als Unterelement von `<tr>`	Abschnitt 14.81
Attribute (Auswahl)	`abbr`, `align`, `axis`, `colspan`, `headers`, `rowspan`, `scope`, `valign`	

Tabelle 6.14 Grundelemente für Tabellen

Sowohl Header- als auch Datenzellen stehen in Zeilencontainern `<tr>`, wobei die Headerzellen automatisch den Datenzellen ihrer Spalte als **Kopfzellen** zugeordnet werden. Textinhalte sind nur innerhalb von Header- und Datenzellencontainern gestattet.

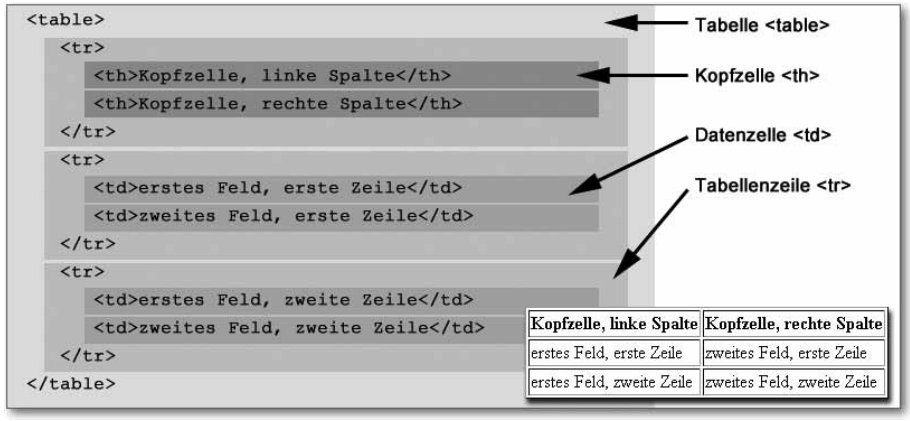

Abbildung 6.14 Grundaufbau einer Tabelle

Eine Tabelle kann jedoch noch weitaus komplexer werden. Da sich die Spalten nur indirekt aus der Zahl der Datenzellen pro Zeile ergeben, wird das Rendering vielspaltiger Tabellen erschwert, da der Browser die reale Spaltenbreite erst ermisst, wenn der jeweils größte Inhalt einer Spalte geladen ist.

Um dies zu erleichtern, existiert die Möglichkeit, zu Beginn der Tabelle die **Spalteneigenschaften** (hauptsächlich die Breite) zu definieren. Hierzu dienen die Elemente `<colgroup>` und `<col>`, wobei letztere die Eigenschaften von Einzelspalten, erstere Gruppen gleichförmiger Spalten festlegen. Ebenfalls zu Beginn der Tabelle, in diesem Fall stets als erstes Kindelement, kann eine **Tabellenüberschrift** `<caption>` stehen, die zentriert über der Tabelle erscheint.

Element	Rolle	Erläuterung
`<caption> ... </caption>`	Erläuternde Überschrift über (Default) bzw. unter der Tabelle; erstes Unterelement von `<table>`	Abschnitt 14.16
`<colgroup> </colgroup>`	Gruppiert mit dem `<col>`-Tag vorgenommene Spaltendefinitionen	Abschnitt 14.21
`<col />`	Definiert die Eigenschaften einer Tabellenspalte als Teil einer Spaltengruppe (`<colgroup>`)	Abschnitt 14.20

Tabelle 6.15 Tabellenelemente für Überschrift und Spaltendefinitionen

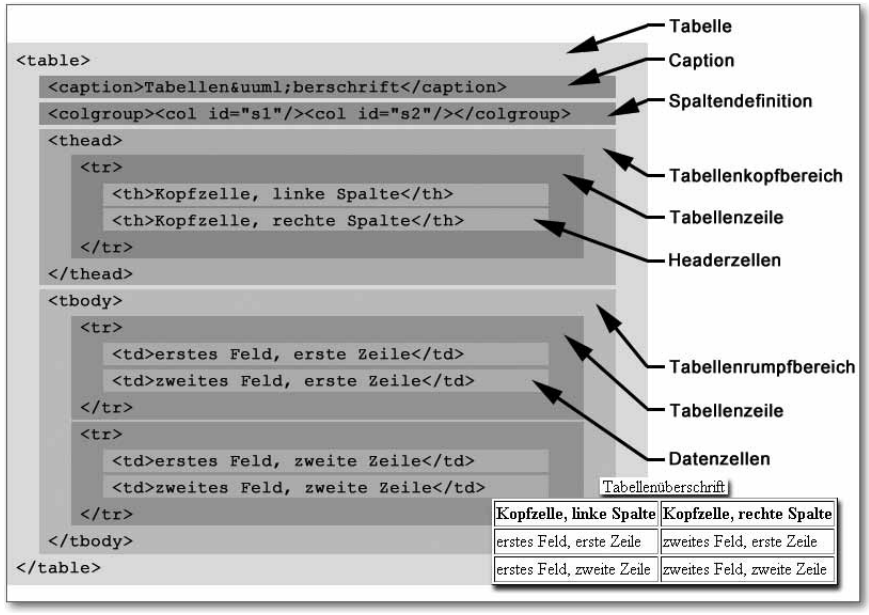

Abbildung 6.15 Aufbau einer komplexeren Tabelle

Weiterhin kann die Tabelle in Bereiche unterteilt werden. Man unterscheidet drei Arten von Bereichen: den **Tabellenkopfbereich** `<thead>`, der stets am Anfang der Tabelle steht (jedoch nach `<caption>` und Spaltendefinitionen), den **Tabellenrumpfbereich** `<tbody>`, der Zeilen `<tr>` mit Datentabellen enthält und den **Tabellenfußbereich** `<tfoot>`, der ebenfalls wieder Zeilen mit Datentabellen enthält. Der Rumpfbereich ist wiederholbar, wodurch man mehrere `<tbody>`-Zeilengruppen erzeugen kann, die bei Bedarf unterschiedlich gestylt werden können (so geschehen für Abbildung 6.13).

Element	Rolle	Erläuterung

`<thead>`
`...`
`</thead>`
	Erklärt die Tabellenzeile(n), die der Container enthält, zum Tabellenkopf	Abschnitt 14.85

`<tfoot>`
`...`
`</tfoot>`
	Erklärt die Tabellenzeile(n), die der Container enthält, zum Tabellenfuß	Abschnitt 14.83

`<tbody>`
`...`
`</tbody>`
	Erklärt die Tabellenzeile(n), die der Container enthält, zum Tabellenkörper	Abschnitt 14.80

Tabelle 6.16 Elemente zur Zeilengruppierung in Tabellen

Nicht per CSS zu erreichen sind Verbindungen zwischen Tabellenfeldern, die entweder zeilen- oder spaltenübergreifend zu definieren sind (auch Kombinationen sind möglich). Hierzu wird in der Ausgangszelle das `colspan`- oder `rowspan`-Attribut gesetzt, dass die Zahl der überspannten Spalten oder Zeilen erhält (die Ausgangsspalte oder -zeile zählt mit).

Abbildung 6.16 Horizontale Verbindung von Tabellenzellen mit colspan

Mit `colspan` und `rowspan` lassen sich beliebig komplexe Tabellengerüste aufbauen. Alternativ können Sie auch Tabellen verschachteln, indem Sie eine innere Tabelle in eine Datenzelle einfügen. Dies war gängige Praxis für Layouttabellen,

dürfte für reine Datentabellen jedoch selten erforderlich sein. Beachten Sie, dass verschachtelte Tabellen den Renderingvorgang bremsen.

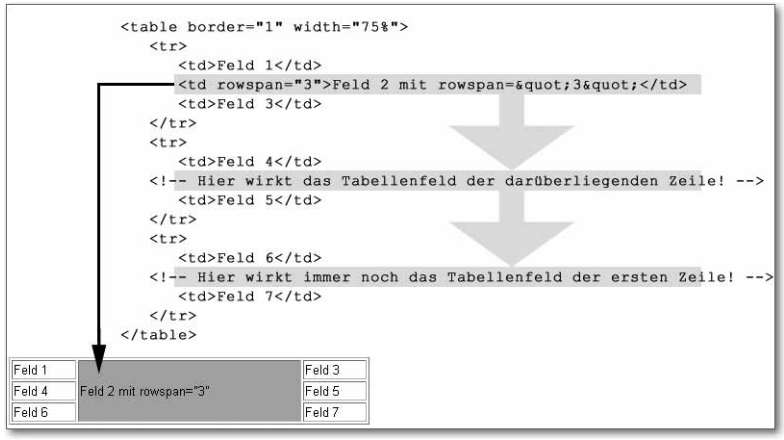

Abbildung 6.17 Vertikale Verbindung von Tabellenzellen mit rowspan

6.6.1 Welche Stylingmöglichkeiten bietet CSS für Tabellen?

Grundsätzlich sind die **Box**- und **Float-Properties** einsetzbar, mit denen Sie auch die Abmessungen der Tabelle festlegen sollten. **Hintergrundfarben** können Sie auf Basis der Tabelle, der Zellen, Zeilen, aber auch der Tabellenbereiche `<thead>` und `<tbody>` definieren.

Wenn Sie **Rahmen** auf Zellen oder Zeilenbasis einsetzen (dies funktioniert nicht für Zeilengruppen), führt dies zu einer möglicherweise **ungewollten Verdoppelung der Rahmenbreite** bei benachbarten Zellen. Dem können Sie mit dem CSS-Property `border-collapse:collapse` abhelfen, dass Sie auf `<table>` anwenden.

Problematisch ist die Anwendung von Ausrichtungsproperties auf **Spaltendefinitionen**. Verwenden Sie stattdessen Klassen für die Datenzellen selbst oder arbeiten Sie auf Zeilenbasis.

6.6.2 Welche HTML-Attribute können für Tabellen verwendet werden?

Müssen Tabellen ohne CSS halbwegs präsentabel wirken, so wird oft ein sichtbares Tabellengitter benötigt. Das `border`-Attribut von `<table>` ist hierfür etwas grobschlächtig. Feinere Möglichkeiten bieten die Attribute `frame` und `rules`, die **Außen- und Innenlinien** getrennt definieren. Am `cellspacing`-Attribut kommen Sie derzeit noch nicht vorbei, da das korrespondierende CSS-Property `border-spacing` noch mangelhaft unterstützt wird. Das Attribut `cellpadding` kann

jedoch durch CSS ersetzt werden (sofern Sie nicht ohne CSS auskommen müssen). Für komplexe Tabellengitter können Sie `colspan` und `rowspan` ausgehend von Header- und Datenzellen einsetzen.

Das **Ausrichten von Inhalten** in der horizontalen ist mit dem CSS-Property `text-align` leicht möglich. Für die vertikale Ausrichtung ist es durchaus akzeptabel, das `valign`-Attribut weiterhin einzusetzen.

Für **barrierefreie Tabellen** sollten Sie über das `summary`-Attribut von `<table>` eine inhaltliche Beschreibung der Tabelle geben. Auch die Attribute `axis` und `scope` der Daten- und Headerzellen fallen in diese Kategorie.

6.7 Formulare

Formulare sind überall dort gefragt, wo Informationen vom Nutzer eingegeben werden sollen, oder gar Dateien von dessen lokalen Rechner auf den Webserver gelangen sollen. Formulare sind, wie Textabsätze, Listen und Tabellen ebenfalls Blockelemente, nehmen also den zur Verfügung stehenden Raum stets in voller Breite ein. Der **Grundaufbau** ist verhältnismäßig einfach – für den Anfang reichen zwei Elemente, wenn das Formular barrierefrei werden soll, kommt noch ein drittes hinzu. Sie benötigen den `<form>`-Container selbst, der alle Bestandteile des Formulars beinhaltet und das `<input>`-Element, ein leeres Element, dessen Erscheinungsbild vom Wert seines `type`-Attributs abhängt.

Input-Typ	Wirkung	Input-Typ	Wirkung
`text`	Texteingabefeld	`password`	Passwortfeld
`radio`	Radiobutton	`checkbox`	Checkboxfeld
`hidden`	unsichtbares Feld	`file`	Dateiuploadfeld
`image`	Grafik als Submit	`button`	neutraler Button
`submit`	Submit-Button	`reset`	Reset-Button

Tabelle 6.17 Die zehn Erscheinungsbilder des `<input>`-Elements

Für den **Formular-Container** werden zwei bis drei Attribute benötigt – vorgeschrieben ist jedoch lediglich das `action`-Attribut. Letzteres nennt das **Ziel des Datenversands**, also die Anwendung, die Seite oder das Script, das die Daten auswerten soll. Zusätzlich kann mit dem `method`-Attribut die Art des Datenversands festgelegt werden. Geben Sie `method="get"` an, so werden die Formulardaten als Querystring an den URL des `action`-Attributs angehängt. Für `method="post"` erfolgt die **Datenübertragung** etwas weniger offen als Teil des HTTP-Headers.

Die **Eingabefelder** benötigen, ungeachtet ihres Typs ein `name`-Attribut, das für den Datenversand gebraucht wird (teilweise wird *parallel* ein `id`-Attribut eingesetzt – geben Sie beiden Attributen in diesem Fall den gleichen Wert). Bestimmte Eingabefelder arbeiten im Verbund, sogenannten Arrays. Dies betrifft **Radiobuttons** und **Checkboxen**. Die Elemente einer solchen Gruppe benötigen, um zusammenzuarbeiten, denselben Wert des `name`-Attributs.

Besteht keine Eingabemöglichkeit in das Feld, müssen die zu übermittelnden Daten anders bestimmt werden – hierzu dient das `value`-Attribut von `<input>`, das bei für Submit- und Reset-Buttons auch die **Buttonaufschrift** bestimmt. (Da die Werte der Submit-Buttons meist nicht verschickt werden müssen, kann für diese das `name`-Attribut entfallen.) Bei **Eingabearrays** wird nur der `value` des tatsächlich angewählten Elements übertragen.

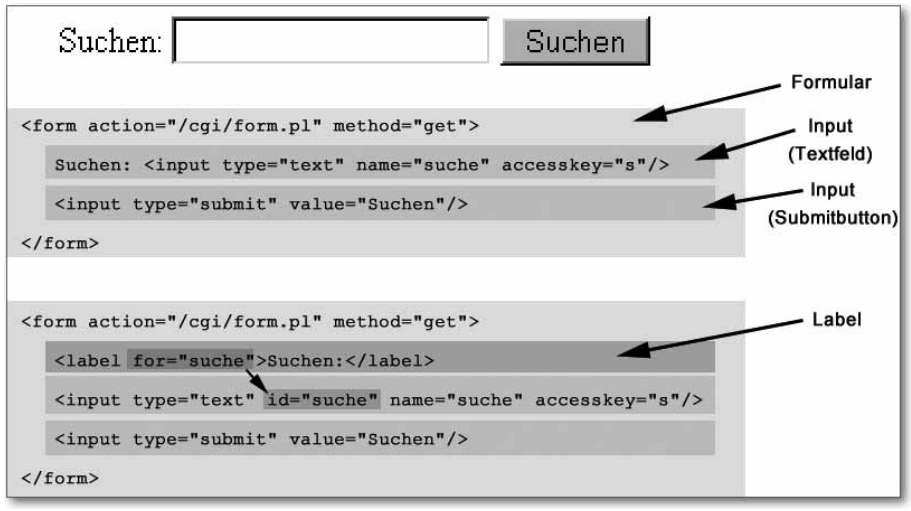

Abbildung 6.18 Einfaches Formular, unten mit hinzugefügtem <label>

Ein Formular gewinnt an **Barrierefreiheit**, wenn die Feldbeschriftungen über `<label>`-Container mit »ihren« Eingabefeldern logisch verknüpft werden. Dies geschieht über eine `ID-IDREF`-Beziehung (siehe Abbildung 6.18, unten). Eine weitere Verbesserung der Barrierefreiheit erhalten Sie durch Beifügung von `accesskey`-Attributen an `<input>`-Elemente, die so über Tastenkürzel ansprechbar werden.

Element	Rolle	Erläuterung
`<form> ... </form>`	Bildet einen Container für Formularelemente; bestimmt über Attribute das Formularverhalten	Abschnitt 14.33
Attribute (Auswahl)	**`action`**, `enctype`, `method`	
`<input />`	Erzeugt verschiedene Arten von Inputelementen, deren Form über das `type`-Attribut bestimmt wird	Abschnitt 14.43
Attribute (Auswahl)	`accesskey`, `checked`, `disabled`, `maxlength`, `name`, `size`, `tabindex`, `type`, `value`	
`<label> ... <label>`	Erzeugt zu einem Formularelement über eine `IDREF-ID`-Beziehung eine klickbare Beschriftung	Abschnitt 14.47
Attribute	`accesskey`, `for`	

Tabelle 6.18 Grundelemente des HTML-Formulars

Spätestens wenn Sie per `<input type="file"/>` einen **Dateiupload** ermöglichen wollen, müssen Sie ein **drittes Attribut** für `<form>` einfügen – in diesem Falle müssen Sie *zwingend* `enctype="multipart/form-data"` angeben, sonst können keine Dateien übertragen werden. Auch *muss* die `POST`-Methode zum Datenversand gewählt sein (Abbildung 6.19).

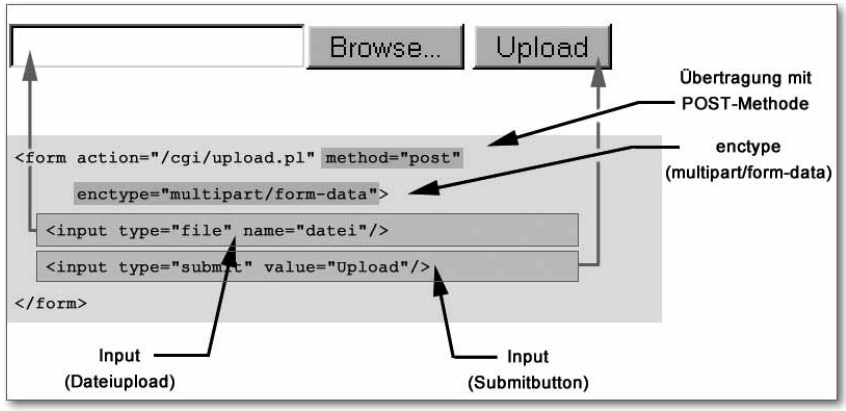

Abbildung 6.19 Formular zum Dateiupload

Neben den einfachen einzeiligen Texteingabefeldern `<input type="text"/>` existiert auch ein Container-Element `<textarea>`, das für größere Textmengen konzipiert ist und mehrzeilig ausgelegt werden kann. Sofern Sie Schaltbuttons mit gestalteter Oberfläche benötigen (der Fläche können auch Grafiken hinzugefügt werden), können Sie auf den `<button>`-Container zurückgreifen.

Element	Rolle	Erläuterung
`<button> ... </button>`	Definiert einen Formularbutton; Containerinhalt wird auf Klickfläche dagestellt	Abschnitt 14.15
Attribute (Auswahl)	`accesskey`, `name`, `tabindex`, `type`, `value`	
`<textarea> ... </textarea>`	Erzeugt ein Texteingabefeld; Attribute legen Abmessungen fest	Abschnitt 14.82
Attribute (Auswahl)	`accesskey`, **`cols`**, `name`, **`rows`**, `tabindex`	

Tabelle 6.19 Formularelemente zur Generierung gestalteter Buttons und Texteingabeflächen

Neben den einfachen Typen von Inputfeldern stehen auch komplexere Eingabefelder zur Verfügung. Das `<select>`-Feld enthält eine Reihe von Optionen, die einzeln, oder (sofern bei `<select>` das Attribut `multiple` gesetzt ist) auch mehrere gewählt werden können. Da auch das `<select>`-Element zu den Eingabefeldern zählt, erhält es wieder ein `name`-Attribut (Abbildung 6.20).

Element	Rolle	Erläuterung
`<select> ... </select>`	Selectbox; dient als Container für Optionselemente	
Attribute (Auswahl)	`multiple`, `name`, `size`, `tabindex`	
`<option> ... </option>`	Erzeugt als Kindelement des Select-Containers dessen einzelne Wahloptionen	Abschnitt 14.62
Attribute (Auswahl)	`disabled`, `selected`, `value`	
`<optgroup> ... </optgroup>`	Definiert eine zusammengehörende Gruppe von Optionselementen einer Selectbox	Abschnitt 14.61
Attribute	`disabled`, `label`	

Tabelle 6.20 Elemente für Auswahlfelder und Combo-Boxen

Einzelne Optionen können mit dem `selected`-Attribut vorselektiert (ohne `multiple` wird jedoch nur eine davon berücksichtigt) oder mit `disabled` stillgelegt werden. Ist kein `value`-Attribut gesetzt, so wird der Inhalt des `<option>`-Containers als Daten verschickt.

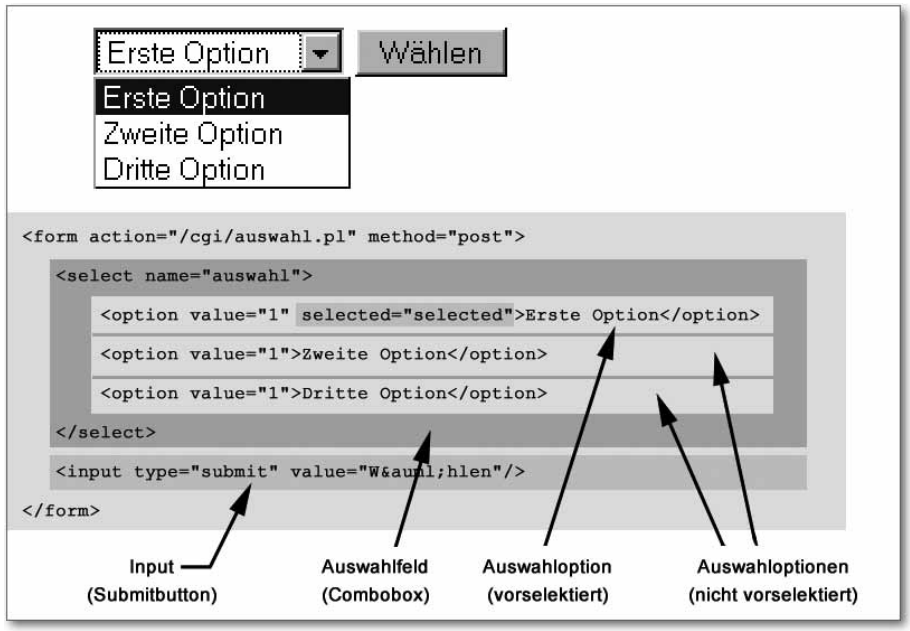

Abbildung 6.20 Formular mit Auswahlfeld <select>

Das Element `<fieldset>` dient zum **Gruppieren von Formularelementen**, wobei in diesem Fall eine logische Gruppierung und keine funktionale (wie bei der Bildung eines Buttonarrays) gemeint ist. Defaultmäßig werden Feldgruppen mit umgebenden Rahmen versehen. Mit `<legend>` kann einer Gruppe eine **Überschrift** zugewiesen werden. Auf die Funktionalität des Formulars haben Feldgruppen jedoch keinen Einfluss.

Element	Rolle	Erläuterung
`<fieldset> ... </fieldset>`	Definiert eine zusammengehörende Gruppe von Inputelementen eines Formulars	Abschnitt 14.31
`<legend> ... </legend>`	Definiert eine Überschrift einer Fieldsetgruppe	Abschnitt 14.48

Tabelle 6.21 Elemente zum Gruppieren von Formularfeldern

Problematisch bei der **optischen Präsentation** von Formularen ist, dass es sich bei allen Formulareingabefeldern um **Inlineelemente** handelt. Sie können sich damit behelfen, eine **Tabelle** zum Ausrichten der Felder und Beschriftungen einzusetzen, was eine sichere und bewährte (wenn auch inzwischen etwas verpönte) Methode darstellt. Sie können auch mit dem **Zeilenumbruchbefehl** `<br/>` arbeiten, um eine Zeilenstruktur zu erzielen.

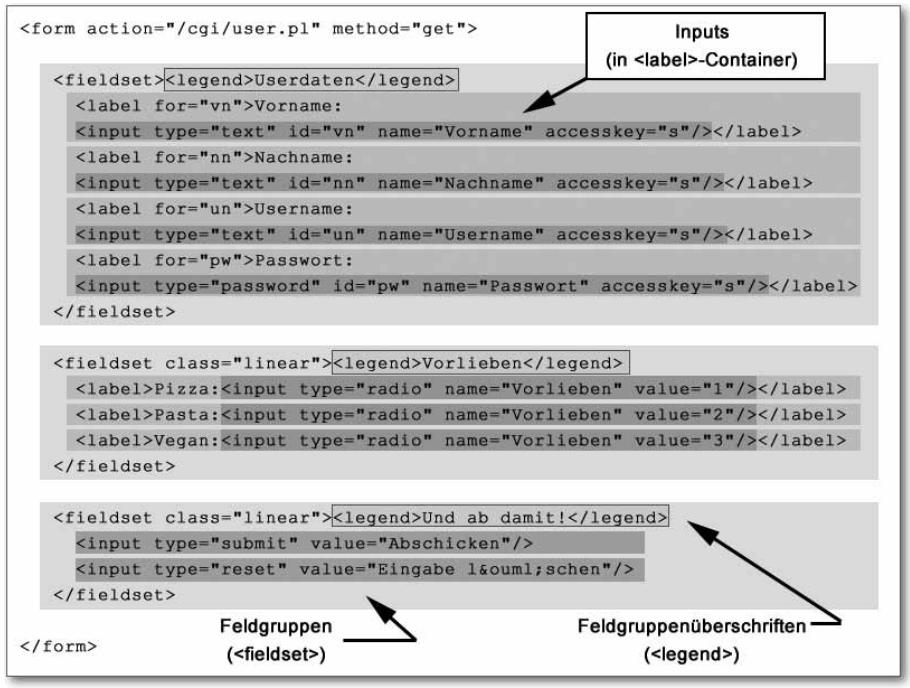

Abbildung 6.21 Formular mit Feldgruppen und -überschriften

Eine gewisse Gestaltungsbasis stellt auch das Blockelement `<fieldset>` dar. Sie können per CSS eine Gesamtbreite für das Formular vergeben und die Feldgruppenbereiche gezielt stylen.

```
<form action="/cgi/user.pl" method="get">
<fieldset><legend>Userdaten</legend>
  <label for="vn">Vorname:
  <input type="text" id="vn" name="Vorname" accesskey="s"/>
  </label>
  <label for="nn">Nachname:
  <input type="text" id="nn" name="Nachname" accesskey="s"/>
  </label>
  <label for="un">Username:
  <input type="text" id="un" name="Username" accesskey="s"/>
  </label>
  <label for="pw">Passwort:
  <input type="password" id="pw" name="Passwort" accesskey="s"/>
  </label>
</fieldset>
<fieldset class="linear"><legend>Vorlieben</legend>
  <label>Pizza:
```

```
  <input type="radio" name="Vorlieben" value="1"/>
  </label>
  <label>Pasta:
  <input type="radio" name="Vorlieben" value="1"/>
  </label>
  <label>Vegan:
  <input type="radio" name="Vorlieben" value="1"/>
  </label>
</fieldset>
<fieldset class="linear"><legend>Und ab damit!</legend>
  <input type="submit" value="Abschicken"/>
  <input type="reset" value="Eingabe l&ouml;schen"/>
</fieldset>
</form>
```

Listing 6.2 Quelltext zu Abbildung 6.21

Für das Formular in Abbildung 6.22 wurden dem `<label>`-Container Block-eigenschaften zugewiesen (`display:block`), was die Felder in einzelne Zeilen gruppiert. Innerhalb der beiden unteren Feldgruppen wurde dies durch einen höherrangigen CSS-Selektor wiederrufen, um die Radiobuttons nebeneinander zu platzieren.

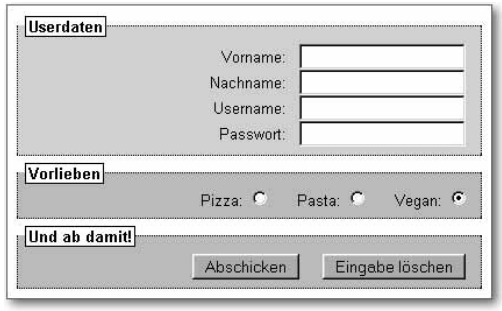

Abbildung 6.22 Umsetzung mit CSS (kapitel_06/formulare.html))

Folgendes CSS wurde in diesem Fall eingesetzt:

```
form      { width:400px;
            font-family:arial,helvetica,sans-serif;
            font-size:0.8em;}
fieldset  { border:1px dotted #000;
            padding:0.5em 0; margin:0.5em 0;
            background-color:#ddd;}
label     { display:block;
            margin-left:1em;
            text-align:right; }
```

```
legend    { margin-left:0.5em;
            font-weight:bold;
            background-color:#fff;
            border:1px solid #000;}
input     { margin:0 1em 0 0.5em; }

/* innerhalb dieser Gruppen bleibt <label> inline: */

fieldset.linear {text-align:right; background-color:#ccc;}
fieldset.linear label {display:inline;}
```

Listing 6.3 CSS zu kapitel_06/formulare.html

6.8 Zusammenfassung

Was Sie aus diesem Kapitel mitnehmen sollten:

- Die wichtigsten Elemente zur Strukturierung von Texten sind Überschriften und Textabsatzcontainer. Überschriften sollten in hierarchischer Reihenfolge verwendet werden.
- Für die Gestaltung innerhalb von Fließtexten existieren Inlineelemente, die ihre Inhalte nach logischen und physischen Gesichtspunkten auszeichnen
- Auf rein präsentativen Markup sollten Sie zugunsten von CSS verzichten.
- Hyperlinks sind ebenfalls Inlineelemente. Sie besitzen eine Defaultpräsentation, können jedoch mit CSS beliebig gestaltet werden.
- Hyperlinks in Fließtext sollten stets auf den ersten Blick als Hyperlinks erkennbar bleiben.
- Mittels Verweisen können Bilder und andere Medien in HTML-Dokumente eingebettet werden.
- Listen existieren din drei Formen, von denen die ungeordnete und die geordnete Liste die meist verwendeten darstellen.
- Tabellen sollten ausschließlich zur Präsentation von tabellierten Daten eingesetzt werden.
- Formulare dienen zum Übertragen von Daten vom Client zum Server. Sie können mit CSS beliebig präsentiert werden.

Wie es weitergeht:

Im folgenden Kapitel wird die Gestaltung von Navigationen auf der Basis ungeordneter Listen erläutert. Vertieft werden darüber hinaus einige Aspekte des Seitenlayouts mit CSS.

In diesem Kapitel erfahren Sie zunächst ein paar theoretische Grundlagen zum Entwurf von Webnavigationen. Im Anschluss werden exemplarische, vielseitig anwendbare und obendrein barrierefreie Navigationslayouts mit CSS auf Grundlage von Listen demonstriert.

7 Navigationsgestaltung mit CSS

Die Aufgabe der Navigation steht im Zentrum der Website – sie dient dazu, den Nutzer zu den gewünschten Informationen zu führen, ihm Orientierung zu bieten und vielleicht überhaupt erst auf das Vorhandensein bestimmter Inhalte aufmerksam zu machen. Dementsprechend wichtig sind ein paar Vorüberlegungen bezüglich Gestaltung und Aufbau einer solchen Navigation.

7.1 Vorüberlegungen

Eine Website ist – abstrakt betrachtet – ein Hypertextsystem, das aus einem Netzwerk verknüpfter Dokumente besteht. Die Struktur eines solchen Systems ist nicht von außen ersichtlich, daher muss die Navigation neben ihrer eigentlichen Funktion auch für die notwendige Orientierung des Nutzers sorgen.

Stets muss die Sitenavigation auf einige Grundfragestellungen Antwort geben können. Diese betreffen die *Orientierung* und die *Bewegung* (Navigation) in der Informationsstruktur einer Site, worunter Folgendes zu verstehen ist:[1]

Orientierungsfragen	Navigationsfragen
Wo bin ich?	Wohin komme ich von hier aus?
Wo war ich schon?	Wie komme ich zurück?
Was ist alles zu finden?	Wie gelange ich von A nach B?
Habe ich alles gefunden?	Wie verfolge ich meinen Weg?
Was ist für mich relevant?	Wo geht es hier raus?

Tabelle 7.1 Grundfragen, die eine Webnavigation beantworten muss

1 In Anlehnung an: L. Rosenfeld/P. Morville: *Information Architecture for the World Wide Web*, O'Reilly 1998 (engl.).

7.1.1 Grundprinzipien der Navigationsgestaltung

An dieser Stelle werden auch Aspekte der Usability berührt, also der Nutzerzufriedenheit. Das Layout eines zusammengehörenden Webangebotes sollte, um gute Usability zu bieten, in einem einheitlichen Stil gehalten sein. Unverständlichkeit und Chaos werden eher selten honoriert, auch wenn sie aus der Motivation kreativen Designs heraus entstehen. Selbstverständlich steht es einem Webdesigner frei, die folgenden Grundregeln unterschiedlich zu gewichten oder ganz zu missachten. Folgende Grundprinzipien haben sich bewährt:

- **Trennung von Navigation und Inhalt**
  Die Navigationselemente sollten außerhalb des Texts in einer Menüleiste zusammengefasst werden, die oben oder links am Rand des Bildschirms eingeblendet wird. (Die Anordnung auf der rechten Seite ist problematisch, da sie der gewohnten Leserichtung entgegensteht.)
- **Konsistenz in Form und Position**
  Gleiche Form und Anordnung von Navigationselementen auf allen Seiten. Der Nutzer sollte sich nicht ständig einer veränderten Optik oder Position der Navigation gegenüber sehen. Für eine vorgeschaltete Splashseite (sofern man dies für nötig hält) kann man hiervon abweichen.
- **Verständlichkeit der Aussage**
  Linktexte sollten verständlich und aussagefähig sein. Beispielsweise ist anstelle von »Weiter? Hier klicken!« der Linktext »Zur Vertiefung zum Thema XY« sicherlich die bessere Wahl.
- **Konsistenz in der Nomenklatur**
  Links, Schalter und Buttons sollen auf auf allen Seiten einheitlich beschriftet sein, wenn sie das gleiche Ziel oder die gleiche Aufgabe haben.
- **Berücksichtigung üblicher Normen**
  Vermeiden Sie ungebräuchliche Abkürzungen und benennen Sie die Funktionen so, dass ihr Zweck klar ist – wenn sich z. B. hinter einem Link funktional ein »Warenkorb« verbirgt, sollte er auch so heißen.
- **Keine Links vortäuschen**
  Unterstrichene blaue Schrift auf weißem Hintergrund sollte Hyperlinks vorbehalten bleiben. Im Fließtext sollten Sie weder die Präsentation von Links völlig verschleiern noch Präsentationen für andere Textbereiche verwenden, die wie Hyperlinks aussehen, ohne es zu sein.
- **Orientierungspunkt bieten**
  Stellen Sie einen »Sie befinden sich hier«-Hinweis zur Verfügung, der die augenblickliche Position in der Sitehierarchie verdeutlicht. Dies wird oft in Form einer so genannten »Breadcrumb-Navigation« realisiert, kann aber auch

innerhalb der Navigation selbst geschehen, indem der »aktuelle« Navigationspunkt hervorgehoben wird (z. B. durch Farbänderung). Beachten Sie aber, dass Letzteres sehbehinderten Nutzern nur begrenzt hilft.

Beschränken Sie sich bei der Planung nicht allein auf die Navigation, sondern sortieren Sie Ihre Inhalte im Vorfeld. Eine »gute« Navigation in einem Webangebot setzt eine gute **inhaltliche Strukturierung** der angebotenen Informationen voraus. Sind diese nicht fachlich sinnvoll gegliedert und auf verschiedene Dokumente verteilt, wird selbst die beste Navigation als chaotisch empfunden.

7.1.2 Arten der Navigation

Dieses Kapitel beschäftigt sich mit der Hauptnavigation der Website, mit der es jedoch in der Regel nicht getan ist. Louis Rosenfeld und Peter Morville unterscheiden in ihrem empfehlenswerten Werk »Information Architecture for the World Wide Web« (O'Reilly 1998) folgende vier Typen von Navigationssystemen:

- **Hierarchische Navigation** (Primäre oder Hauptnavigation)
- **Lokale Navigation** (Sekundäre oder Bereichsnavigation)
- **Globale Navigation** (Präsenz- oder Service-Navigation)
- **»Ad Hoc«-Navigation** (Spontannavigation im Inhaltsbereich)

Hierarchische Navigation

Obwohl die Grenzen zwischen diesen abstrakten Navigationstypen fließend sind, lohnt eine kurze Diskussion. Der Begriff »hierarchische« Navigation drückt aus, dass hier die *Grundstruktur* der Website abgebildet wird. Dies ist die Aufgabe der **Hauptnavigation**.

Außer bei Sites mit geringem Umfang wird hier selten die vollständige Hierarchie gezeigt, sondern es wird eine Beschränkung auf die Hauptbereiche vorgenommen. Als Faustregel gelten sieben Hauptnavigationspunkte als Obergrenze des auf einen Blick Erfassbaren. Zusätzlich sollten Sie bei vertikal angeordneter Navigation die wichtigsten Menüpunkte oben platzieren, damit sie zuverlässig *im Sichtbereich* erscheinen. Üblicherweise ist auch der Home-Link Teil dieser Navigation, obwohl er eigentlich keinen »Bereich« repräsentiert (Sie könnten ihn daher ebenso der globalen Navigation zuordnen).

Lokale Navigation

Die Aufgabe der Abbildung der tiefer liegenden Hierarchie wird von der lokalen Navigation übernommen, die, weil sie innerhalb des aktuellen Teilbereichs navi-

gieren hilft, auch als **Bereichsnavigation** bezeichnet wird. Sie kann als eigenständiges Menü abseits der Hauptnavigation ausgebildet oder optisch in dieses integriert sein (in Form ausklappender oder eingeblendeter Untermenüs). Hierbei muss die hierarchische Unterordnung unter die Hauptnavigation optisch deutlich werden. Ähnlich wie bei der Hauptnavigation umfasst die Bereichsnavigation nicht unbedingt alle Seiten des Bereiches. Um die Anzahl der Menüpunkte nicht unübersichtlich groß werden zu lassen, weichen Sie für weniger frequentierte Seiten besser auf eine »Ad Hoc«-Navigation aus.

Typische Beispiele:

- Submenüs für den aktuellen Bereich
- Themenübersichten
- Seitennavigation bei längeren, unterteilten Artikeln, beispielsweise in der Form `(1|2|3|4|5|6|7|8|9|10)` als horizontale Hilfsnavigation (die aktuelle Postion kann durch Fettauszeichnung oder ähnlich hervorgehoben werden (hier ist es Seite 3).

Globale Navigation

Die globale Navigation kann ebenfalls in die Hauptnavigation integriert sein. In der Regel werden diese Funktionen jedoch separat angeboten und, da sie stets unmittelbar verfügbar sein sollen, am oberen Seitenrand oder zumindest im ständigen Sichtbereich platziert.

Zur **globalen Navigation** in diesem Sinne gehören Kontakt- und Impressum-Links, Hilfe- und FAQ-Funktionen, aber auch Sitemap-Link und das Suchfeld einer siteinternen Suchfunktion. Da dies Funktionalitäten sind, die nützliche, allgemeine Dienste leisten, bezeichnet man sie auch als »Service-Navigation« und bezeichnet den Seitenbereich, den sie einnehmen als **Servicebereich**. Die globalen Navigationspunkte können jedoch auch auf mehrere räumlich getrennte Bereiche der Seite verteilt sein.

»Ad Hoc«-Navigation

Ebenfalls zur Navigation in der Website dienen jene Hyperlinks, die im Inhaltsbereich der Seite platziert sind. Dies kann den Grund haben, dass sie »spontan« verwendet werden sollen, auf Unterseiten führen, die hierarchisch soweit untergeordnet sind, dass ihre Aufnahme in die Bereichsnavigation diese unnötig aufblähen würde, oder dass sie auf externe Ziele oder Dateidownloads gerichtet sind.

7.2 Navigationsmarkup

Hinter jedem Navigationsmenü steht notwendigerweise HTML-Quellcode. Im Zentrum steht wieder der Hyperlinkcontainer `<a>`. (Auf Flash-basierte Navigation wird hier nicht eingegangen – sie mag trotz ihrer Barriereproblematik hier und da angebracht sein, verlässt aber den Themenbereich dieses Buches.)

Da es sich um einen Inlinecontainer handelt, muss die optische Anordnung mit Hilfe weiterer Elemente erfolgen, sobald die Links untereinander platziert werden sollen. Sollen die Links in Form einer horizontal orientierten Navigation nebeneinander stehen, ist dies nicht zwingend erforderlich.

7.2.1 Horizontale Navigation mit Trennsymbolen

Für ein horizontales Menü können Sie Hyperlinkcontainer einfach nebeneinander platzieren, sofern Sie darauf achten, dass zwischen den Linkcontainern Trennsymbole gesetzt sind. Sonst fällt es andernfalls Screenreadern schwer, die einzelnen Linktexte auseinanderzuhalten (dies gilt als Barriereproblem).

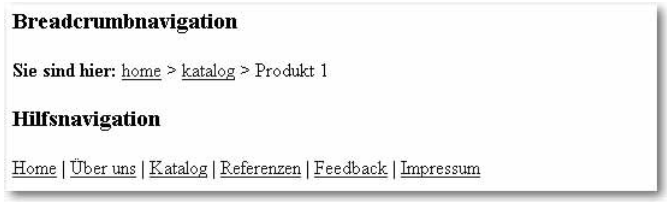

Abbildung 7.1 Beispiel für Breadcrumb- und Hilfsnavigation

Üblich ist dieses Verfahren für **Breadcrumb-Navigation** oder für Hilfsnavigationen im Seitenfußbereich. In der Regel wird ein umliegender Container existieren, entweder ein `<div>`-Element mit ID oder (weniger günstig) eine Tabellenzelle im Rahmen eines Tabellenlayouts.

Beispiel für eine Breadcrumb-Navigation

```
<div id="breadcrumb"><strong>Sie sind hier: </strong>
   <a href="index.html">home</a> &gt;
   <a href="katalog.html">katalog</a> &gt;
    Produkt 1
</div>
```

Die spitzen Klammern verdeutlichen den Hierarchiepfad. Sie werden durch Entities repräsentiert. Die aktuelle Seite taucht am Ende der Breadcrumb-Navigation auf, braucht jedoch keinen Link zu besitzen. Der Nutzer kann auf die beiden

übergeordneten Hierarchiestufen »Katalog« und »Home« direkt zugreifen (siehe Abbildung 7.1, oben).

Beispiel für eine Hilfsnavigation

```
<div id="bottomnavi">
   <a href="index.html">Home</a> |
   <a href="wir.html">&Uuml;ber uns</a> |
   <a href="katalog.html">Katalog</a> |
   <a href="referenzen.html">Referenzen</a> |
   <a href="feedback.html">Feedback</a> |
   <a href="Impressum.html">Impressum</a>
</div>
```

Eine **Hilfsnavigation** (siehe Abbildung 7.1, unten) spiegelt die wichtigen Teile der Hauptnavigation wider. Sie wird meist am unteren Seitenrand platziert und steht zur Verfügung, wenn die Hauptnavigation aus dem Sichtbereich gescrollt ist. Gegebenenfalls können (wie hier der Impressum-Link) auch Teile der globalen Navigation integriert werden. Als Trennsymbol fungiert üblicherweise der senkrechte Strich (Pipe-Symbol), da hier Hierarchiegleichheit vorliegt. Selbstverständlich können auch andere Symbole wie Klammern oder Grafiken eingesetzt werden. Trennen Sie die Navigationslinks aber auch in diesem Fall nicht allein durch Leerzeichen (auch nicht durch eine Folge geschützter Leerzeichen ` `). Vor den ersten und nach dem letzten Link braucht kein Trennsymbol stehen.

7.2.2 Vertikale Navigation mit Zeilenumbrüchen

Um Hyperlinks einfach übereinander zu platzieren, genügen Zeilenumbruchsbefehle `<br/>`. Die Präsentationsmöglichkeiten sind jedoch rudimentär – sie beschränken sich auf das Styling des umliegenden Blockcontainers und die Festlegung von Schriftart bzw. -farbe und Zeilenhöhe. In manchen Fällen mag dies aber ausreichen.

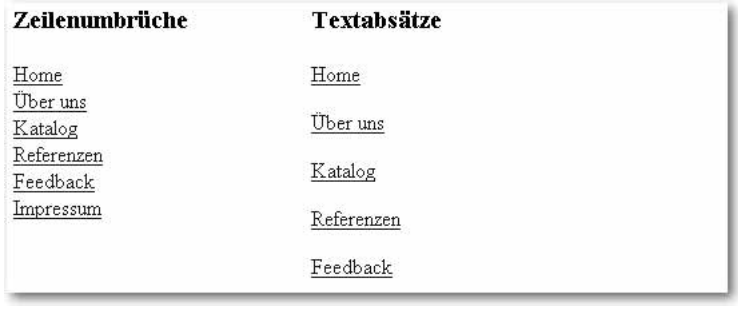

Abbildung 7.2 Navigationen mit Zeilenumbrüchen und Textabsätzen

Beispiel für eine vertikale Navigation mit Zeilenumbrüchen

```
<div id="hauptnavi">
   <a href="index.html">Home</a> <br/>
   <a href="wir.html">&Uuml;ber uns</a> <br/>
   <a href="katalog.html">Katalog</a> <br/>
   <a href="referenzen.html">Referenzen</a> <br/>
   <a href="feedback.html">Feedback</a> <br/>
   <a href="Impressum.html">Impressum</a>
</div>
```

Der Nachteil ist, dass per CSS ohne weitere Manipulation der Containereigenschaften von `<a>` nur die Hintergrundfarbe des gesamten Bereichs befriedigend zu gestalten ist. Würden Sie den Links selbst individuelle Farben geben, so wären Sie auf die jeweilige Textfläche beschränkt, was bei ungleicher Wortlänge ein »zackiges« Gesamtbild ergäbe. Zudem würden die Farbflächen der Links einander berühren.

Abbildung 7.3 Auf Zeilenumbruch basierende Navigation, leicht gestylt

Das für Abbildung 7.3 verwendete CSS sieht folgendermaßen aus:

```
#hauptnavi {width:170px;
            background-color:#eee;
            border:1px dotted #aaa;
            padding:1em;
            line-height:160%; }
```

Hinweis: Der Internet Explorer bis 6.0 stellt gepunktete Rahmen mit 1 px Breite leider im falschen Stil dar, nämlich gestrichelt (erst ab 2 px Breite werden Rahmen tatsächlich gepunktet gezeigt). Für 1 px breite Rahmen müssen Sie daher für vollständige Darstellungsübereinstimmung auf die Stile `solid` oder `dashed` ausweichen. **[«]**

7.2.3 Vertikale Navigation mit Textabsätzen

Bessere Möglichkeiten bietet die Gestaltung mit Textabsätzen (Sie können auch `<div>`-Container einsetzen). Die Links sind nun optisch und semantisch besser voneinander getrennt und auch die Absatzcontainer können individuell gestylt werden, um beispielsweise die aktuelle Position hervorzuheben.

```
<div id="hauptnavi">
   <p class="current"><a href="index.html">Home</a></p>
   <p><a href="wir.html">&Uuml;ber uns</a></p>
   <p><a href="katalog.html">Katalog</a></p>
   <p><a href="referenzen.html">Referenzen</a></p>
   <p><a href="feedback.html">Feedback</a></p>
   <p><a href="Impressum.html">Impressum</a></p>
</div>
```

Ein Nachteil ist, dass nur der Linktext als **aktive Fläche** dient, obwohl durch CSS-Präsentation des `<p>`-Containers der Anschein eines Buttons gegeben werden kann. Hierzu existiert eine Abhilfe, die in Folge beschrieben wird. Sie ist auch zusammen mit der folgenden Grundstruktur anwendbar, der listenbasierten Navigation.

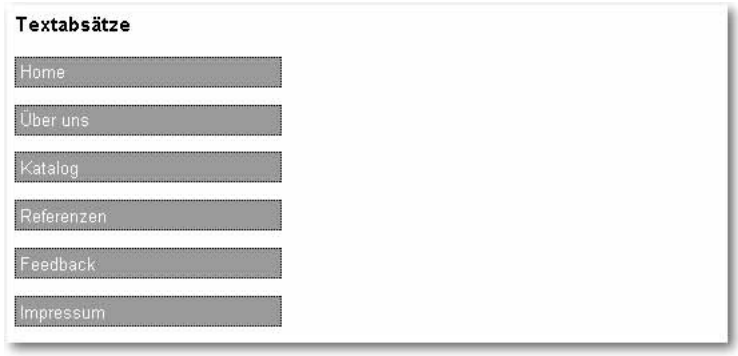

Abbildung 7.4 Textabsatzbasierte Navigation, leicht gestylt

Das für Abbildung 7.4 verwendete CSS sieht folgendermaßen aus:

```
#hauptnavi   { width:200px;}

#hauptnavi p { background-color:#999;
               padding:0.2em;
               border:1px dotted #000; }

#hauptnavi a { color:#fff;
               text-decoration:none;}
```

7.2.4 Vertikale Navigation mit Listen

Die semantisch korrekteste Art eine Navigation umzusetzen, ist eine *ungeordnete Liste* – immerhin handelt es sich ja um eine »Auflistung von Menüpunkten«. Die wenig attraktive Defaultpräsentation solcher Listen wirkt eher abschreckend, jedoch ist über CSS jede beliebige Darstellung möglich.

```
<div id="hauptnavi">
<ul>
   <li><a href="index.html">Home</a></li>
   <li><a href="wir.html">&Uuml;ber uns</a></li>
   <li><a href="katalog.html">Katalog</a></li>
   <li><a href="referenzen.html">Referenzen</a></li>
   <li><a href="feedback.html">Feedback</a></li>
   <li><a href="Impressum.html">Impressum</a></li>
</ul>
</div>
```

Die Aufgabe der `<p>`-Container des vorigen Beispiels wird hier durch den `<li>`-Container übernommen. Der Vorteil dieser Herangehensweise ist die Verdeutlichung der Zusammengehörigkeit der Navigationspunkte. Zudem ist eine Navigation mit Unterhierarchien nun durch *verschachtelte Listen* erzeugbar:

```
<div id="hauptnavi">
<ul>
   <li><a href="index.html">Home</a></li>
   <li><a href="wir.html">&Uuml;ber uns</a></li>
   <li><a href="katalog.html">Katalog</a>
   <!-- Untermenüebene: -->
      <ul>
         <li><a href="produkt1.html">Produkt 1</a></li>
         <li><a href="produkt2.html">Produkt 2</a></li>
         <li><a href="produkt3.html">Produkt 3</a></li>
      </ul>
   </li>
   <li><a href="referenzen.html">Referenzen</a></li>
   <li><a href="feedback.html">Feedback</a></li>
   <li><a href="Impressum.html">Impressum</a></li>
</ul>
</div>
```

Ohne CSS sieht diese Navigation noch nach nichts aus. Bereits mit einfachen Manipulationen wie dem Entfernen der Aufzählungspunkte und der Vergabe von ein paar Rahmen- und Farbanweisungen lässt sich dieser Navigationstyp jedoch angenehm präsentieren. Für einige Feinheiten sind jedoch weitere CSS-Kniffe erforderlich.

Abbildung 7.5 Nüchtern, aber funktional: ungestylte Listen-Navigationen

7.3 Vertikale Navigation

Die Gestaltung einer vertikal angeordneten Navigation auf Basis einer Liste ist einfach, da die äußere Form der Liste dabei erhalten bleibt. Der erste Schritt zu einem ansprechend aussehenden Menü besteht stets im *Deaktivieren* der Bullets. Dies geschieht durch das Property `list-style-type`, das für alle Listen im Navigationscontainer auf den Wert `none` gesetzt wird.

```
div#navigation ul {
      /* Aufz&auml;hlungselemente enfernen: */
       list-style-type:none; }
```

Für Abbildung 7.6 und Abbildung 7.7 wurde der `<div>`-Container mit durchgezogenem und die Liste mit gestricheltem Rahmen markiert.

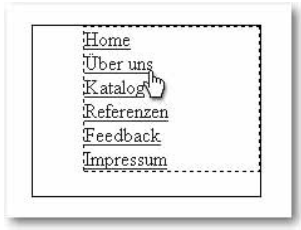

Abbildung 7.6 Liste im Navigationscontainer, noch mit Margin

Nun werden zusätzlich die Margins und Paddings des `<ul>`-Containers und seiner Listenelemente genullt, damit die Liste bündig im `<div>`-Container sitzt (Abbildung 7.7):

```
div#navigation ul, div#navigation li {
       /* Au&szlig;enabstand entfernen: */
       margin:0; padding :0 ; }
```

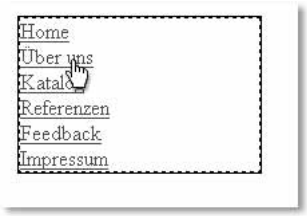

Abbildung 7.7 Die Liste sitzt bündig im Navigationscontainer.

Nun trennen nur noch wenige Handgriffe von einer Ansicht in Buttonform. Das Listenelement `<li>` erhält Hintergrundfarbe, Rahmen, Padding und eine definierte Breite. Damit die so entstehenden Boxen nicht aneinander stoßen, gibt es noch ein kleines Margin nach unten.

```
li {border:1px solid #555;
       padding:5px;
       width:140px;
       margin-bottom:5px;
       background-color: #ccc; }
```

Schließlich wird noch der Linkunterstrich entfernt und die Textfarbe festgelegt.

```
div#navigation a { color:#000;
       text-decoration:none; }
```

Abbildung 7.8 Die fertige Boxpräsentation der Liste

Mag dies vom Layout (sofern man nicht auf unbedingte Originalität besteht) durchaus schon durchgehen, von der Funktionalität ist bei dieser Navigation noch einiges zu verbessern.

7.4 Navigationspräsentation

Neben der nach einem ansprechenden Aussehen gelten für eine Navigation auch funktionale Forderungen. Die Barrierefreiheit verlangt eine große aktive *Linkfläche* (Klickfläche), die Usability eine *Rückmeldung* über den Zustand.

Die nutzbare Klickfläche – Wo ist das Problem?

In den eben gezeigten Beispielen ist die aktive Fläche auf den Textbereich des Links beschränkt (siehe Abbildung 7.9, hier ist der Text zur Verdeutlichung mit einer Hintergrundfarbe hinterlegt), obwohl durch Padding, Hintergrundfarbe und Rahmen der optische Eindruck eines flächigen Buttons hervorgerufen wird. Durch Einfügen einer entsprechend großen (Text-)Grafik anstelle des Linktextes stünde zwar deren gesamte Fläche zu Verfügung. Hiermit würde man sich jedoch andere Probleme einhandeln; die Zugänglichkeit wäre nun für nicht visuell arbeitende User Agents erschwert. Da diese Art des Einsatzes von Grafiken hier ausgeschlossen ist, muss (und kann) eine Lösung mittels CSS gefunden werden.

Abbildung 7.9 Nur der Text ist anklickbar, die restliche Fläche nutzlos.

Die optische Rückmeldung – Geht es ohne JavaScript?

Eine optische Rückmeldung in Form von Mouseover-Effekten bedeutete früher den Einsatz von grafischen Menüs und JavaScript. Dies ist dank der in CSS einsetzbaren so genannten Pseudoselektoren nicht mehr erforderlich (allerdings lassen sich weniger Zustände des Links unterscheiden, als Maus-Events existieren). Es gibt zwar noch einige Fälle, in denen man auf JavaScript angewiesen bleibt, für die verbreitetsten Anforderungen genügt pures CSS.

7.4.1 Das Display-Property – Den Anker zur Fläche machen

Da es sich beim Anker-Container um ein Inlineelement handelt, ist die aktive Fläche auf die Größe seiner Inhalte beschränkt. Grafiken wurden als mögliche (sonst durchaus erlaubte) Inhalte ausgeschlossen. Da auch die Linktexte nicht angetastet

werden können, müssen die Eigenschaften des Containers selbst geändert werden. Hierzu kann das CSS-Property `display` verwendet werden. Folgende Anweisung macht aus dem Inlineelement `<a>` ein Blockelement:

```
a   { display:block; }
```

Nun sollen aber nicht pauschal alle Links des Dokuments zu Blockelementen werden (dies würde im Fließtext Chaos auslösen). Um die Wirkung auf den Navigationscontainer zu begrenzen, wird ein *Kontextselektor* eingesetzt:

```
div#navigation  a  { display:block; }
```

Wie funktioniert ein CSS-Kontextselektor?

Der Kontextselektor ist ein zusammengesetzer Selektor. Er besteht aus mindestens zwei Teilselektoren, bei denen es sich um Typ-, Klassen- oder ID-Selektoren handelt (es gibt noch weitere Möglichkeiten). Der Kontextselektor drückt eine Hierarchie aus, an deren Ende das durch den letzten Teilselektor erfasste Element steht (hier also ein Element vom Typ `<element_c>`).

```
element_a  element_b  element_c
```

Dieses `<element_c>` muss allerdings die Kontextbedingung erfüllen, die darin besteht, dass durch die vorausgehenden Selektoren (in dieser Reihenfolge) dessen übergeordnete Container erfasst werden: Es wird also ein Element `<element_c>` *nur dann* gewählt, wenn er in einem `<element_b>` enthalten ist, das wiederum in einem `<element_a>` enthalten ist.

Ob dazwischen weitere Hierarchiestufen (sprich Elemente) existieren, ist unerheblich. Daher erfasst `div#navigation a` alle `<a>`-Container, die *irgendwo* innerhalb von `<div id="navigation">` stehen.

Weil den Kontextselektor die restliche Hierarchie zwischen `<div id="navigation">` und `<a>` nicht interessiert, ist es unerheblich, ob die Navigation mittels einer Liste oder mit Textabsätzen realisiert wurde. Der Kontextselektor sagt aus, dass alle `<a>`-Container *irgendwo innerhalb* von `<div id="navigation">` als Blockelemente zu behandeln sind. Zur Verdeutlichung wurde für den `<a>`-Container für Abbildung 7.9, Abbildung 7.10 und Abbildung 7.11 eine gelbe Hintergrundfarbe vereinbart.

Zu diesem Zeitpunkt stellt es sich als ungünstig heraus, dass der umgebende Container ein Padding erhalten hat: Man erkennt, dass der Anker-Container die Fläche nicht wie gewollt ausfüllt. Es ist also notwendig, das Padding des umliegenden Containers auf Null zu setzen und stattdessen dem `<a>`-Container zuzuweisen. Nun füllt der Anker die Buttonfläche optisch aus (Abbildung 7.11).

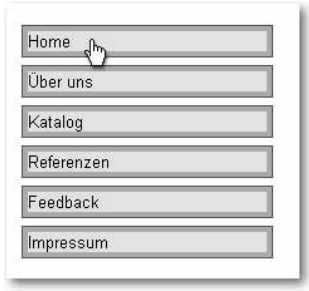

Abbildung 7.10 Die Fläche des <a>-Containers dehnt sich aus.

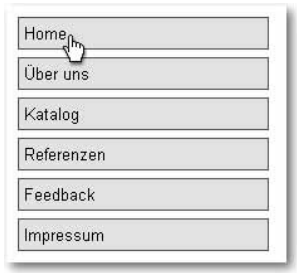

Abbildung 7.11 Der <a>-Container füllt die Buttonfläche vollständig.

Das hier auf den `<a>`-Container angewendete CSS lautet wie folgt:

```
div#navigation a { color:#000;
background-color: #ff0;
                  text-decoration:none;
                  display:block;
                  padding:0.3em; }
```

Nicht anwendbar für Zeilenumbruch-Navigation

Wurden *Zeilenumbruchbefehle* als Basis der Navigation eingesetzt, so verdoppeln sich die Abstände zwischen den Navigationspunkten durch `display:block`. Hier ist der Weg nicht gangbar. Auch das nun eventuell verlockend wirkende Entfernen der Zeilenumbrüche (da `<a>` ein Blockelement ist, scheinen sie ja nicht mehr erforderlich) ist keine brauchbare Option: Ohne CSS stehen die Links wieder als Inlineelemente nebeneinander – nun aber ohne trennenden Zeilenumbruch.

[»] *Hinweis:* Der Internet Explorer beharrt darauf, dass auch für einen Blockanker nur der Textbereich aktiv ist. Dennoch ist die Änderung auch für ihn (im Hinblick auf den gleich folgenden Abschnitt) sinnvoll. In Mozilla/Firefox und Opera dehnt sich der aktive Linkbereich nun auf den ganzen Bereich des umgebenden Containers aus.

7.4.2 CSS-Pseudoklassen – Mouse-Over-Effekte mit CSS

Wenn eine Navigation auf seine Aktion reagiert, fühlt sich ein Nutzer darin bestätigt – aus diesem Grunde besteht das am häufigsten eingesetzte Feature in einem Mouseover-Effekt für die Navigation. Eine Reaktion des Links auf das Überfahren mit der Maus macht deutlich: Dieser Link ist aktiv und kann angeklickt werden. Was für JavaScript »Ereignisse« an einem Element sind, bezeichnet CSS als einen »Zustand« des Elements. Für verschiedene Zustände können eigene Präsentationsanweisungen bereitgestellt werden. Weil kein Element, sondern dessen Zustand ausschlaggebend für das Erfassen durch den Selektor ist, wird dieser als *Pseudoselektor* bezeichnet

Der Zustand des Ankerelements bei Mouseover heißt `"hover"`, woraus sich auch der Name des Pseudoselektor ableitet. Zur Verwendung hängt man ihn einfach an den Typselektor des beobachteten Elements an, hier also an den Ankercontainer `<a>`. Als Verbinder dient der Doppelpunkt:

```
div#navigation a:hover { background-color:#f00;
          font-weight:bold;
          color:#fff; }
```

Auch hier musste die Wirkung auf den Navigationsbereich beschränkt werden, daher wird wieder der ID-Selektor des Bereichs ins Spiel gebracht.

Abbildung 7.12 Bei Mouseover wird die Präsentation des Links geändert.

Für einfache Rollover-Effekte kann also auf JavaScript vollkommen verzichtet werden. Für den Hover-Zustand können sämtliche Parameter verändert werden – es ist also auch das Umschalten zwischen verschiedenen Hintergrundbildern möglich, was sehr radikale Effekte ermöglicht.

Anmerkung: Während die restlichen aktuellen Browser schon länger Hover-Zustände anderer Elemente (beispielsweise von `<p>`-Containern) unterstützen, beharrte der Internet Explorer bis Version 6 darauf, dies nur auf den `<a>`-Container anzuwenden (Lösungen sind mit Scripting möglich, aber nicht Thema dieses Buchs). Internet Explorer 7 unterstützt `:hover` nun allgemein.

7.5 Vertikale verschachtelte Navigation

Sobald Navigationslisten verschachtelt werden, ergibt sich zwanglos eine hierarchische Anordnung, die sich zur Abbildung von Haupt- und Bereichsnavigation eignet. Die verschachtelten Listen müssen jedoch abweichend von den Hauptlisten gestylt werden. Die bisher eingesetzten Selektoren erfassen jedoch *sowohl* Haupt- als auch Unterliste:

```
div#navigation ul {...}
div#navigation ul li {...}
```

Da der Kontextselektor bezüglich des hierarchischen Auftretens der Zielelemente nicht wählerisch ist, gelten beide Regeln auch für die innen liegende Unterliste. Hierfür müssen nun spezifischere Selektoren vereinbart werden:

```
div#navigation ul ul {...}
div#navigation ul ul li {...}
```

Die erste Regel gilt für Listen, die innerhalb einer Liste stehen, erfasst also die Unterliste (aber auch eventuelle, dieser untergeordnete Listen, falls diese existieren). Die zweite Regel erfasst ein Listenelement, wenn ihm zwei Listencontainer übergeordnet sind. Dies gilt für die Listenelemente der inneren, aber nicht für die der äußeren Liste.

Abbildung 7.13 Beispiel für gemischte Haupt- und Bereichsnavigation

Nun soll exemplarisch eine verschachtelte Navigation gestaltet werden. Die Hauptnavigationspunkte sollen als Boxen realisiert werden, die einen Hover-Effekt besitzen, die Unternavigationspunkte hingegen sollen ohne Box als einfache Links gezeigt werden. Als Hover-Effekt soll für sie der Linkunterstrich eingeblendet werden. Hier ein Ausriss aus dem Quelltext:

```
<div id="navigation">
<ul>
   <li><a href="#">Home</a></li>
   <li class="current">
   <a href="wir.html">&Uuml;ber uns</a>
   <!-- Untermenueebene: -->
      <ul>
         <li><a href="#">Abteilung 1</a></li>
         <li><a href="#">Abteilung 2</a></li>
         <li><a href="#">Abteilung 3</a></li>
      </ul>
   </li>
   ...
</ul>
</div>
```

Zunächst werden die Listen, wie inzwischen geläufig, vorbereitet:

```
div#navigation ul { list-style-type:none; }

div#navigation ul, div#navigation  li {
           margin:0;padding:0; }
```

Das Listenelement erhält eine Breite:

```
div#navigation  li { width:150px; }
```

Der Ankercontainer wird als Block definiert. Aufgrund des Kontextselektors gelten diese Regeln jedoch auch (noch) für die Anker der inneren Liste:

```
div#navigation ul li a {  display:block;
           width:120px; color:#000;
           border:1px solid #999;
           background-color: #ccc;
           text-decoration:none;
           padding:1px;
           margin-bottom:5px; }
```

Auch ein Hover-Effekt für den Anker wird vereinbart:

```
   div#navigation ul a:hover {
           background-color: #aaa; }
```

Die Link-Container des untergeordneten Menüs sollen anders gestylt werden. Da der zusammengesetzte Selektor eine höhere Spezifizität besitzt, werden hier wie gewünscht die für die äußeren Ankercontainer geltenden Eigenschaften überschrieben:

```
div#navigation ul ul a {
           display:block;
           text-align:right;
           border:0;
           color:#000;
           background-color: #fff;
           text-decoration:none;
           padding:0:
           margin-bottom:5px; }
```

Er erhält auch einen anderen Hover-Effekt:

```
div#navigation ul ul a:hover {
           text-decoration:underline;
           background-color: #fff; }
```

Ein- und Ausblenden von Unternavigationen

Sofern der Umfang der Unternavigationen nicht gerade riesig ist, mag es ökonomischer sein, auf allen Seiten die gleiche Navigation einzusetzen.

```
<div id="navigation">
<ul>
   <li><a href="#">Home</a></li>
   <li class="current">
   <a href="wir.html">&Uuml;ber uns</a>
   <!-- Untermenueebene: -->
      <ul>
         <li><a href="#">Abteilung 1</a></li>
         <li><a href="#">Abteilung 2</a></li>
         <li><a href="#">Abteilung 3</a></li>
      </ul>
   </li>
   <li><a href="katalog.html">Katalog</a>
   <!-- Untermenueebene: -->
      <ul>
         <li><a href="#">Produkt 1</a></li>
         <li><a href="#">Produkt 2</a></li>
         <li><a href="#">Produkt 3</a></li>
      </ul>
   </li>
```

```
   <li><a href="referenzen.html">Referenzen</a>
   <!-- Untermenueebene: -->
      <ul>
         <li><a href="#">Kunde 1</a></li>
         <li><a href="#">Kunde 2</a></li>
         <li><a href="#">Kunde 3</a></li>
      </ul>
   </li>
   <li><a href="#">Feedback</a></li>
   <li><a href="#">Impressum</a></li>
</ul>
</div>
```

Nur das Untermenü des aktuellen Links wird eingeblendet, alle anderen Untermenüs (die nicht in einem `<li class="current">` eingeschlossen sind) sind ausgeblendet:

```
div#navigation ul li ul         {display:none;}
div#navigation ul li.current ul {display:block;}
```

Hiermit ergeben sich leicht zu variierende hierarchische Menüs, die (sieht man von der Vergabe der Klasse `current` ab) auf jeder Seite gleich sind – allerdings wird jedes Mal nur das Untermenü der aktuellen Seite einblendet.

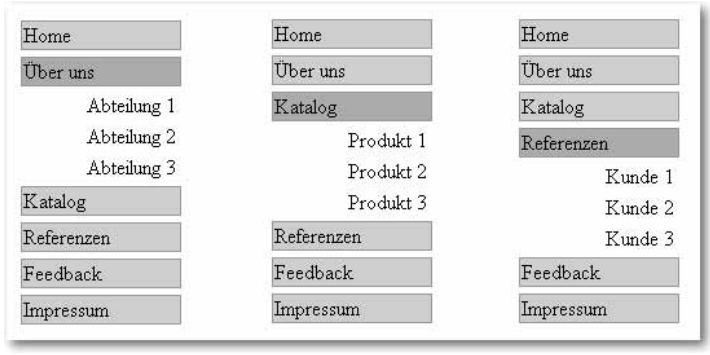

Abbildung 7.14 Drei Varianten eingeblendeter Untermenüs

7.6 Horizontale Navigation

Den Ausgangspunkt bildet eine ungeordnete Liste. Das zugrunde liegende HTML ist daher sehr einfach. Die einzige Besonderheit ist wieder die Stylesheetklasse `.current`, die demjenigen Listenelement hinzugefügt wird, welches den aktuellen Menüpunkt repräsentiert. Auch der umliegende `<div>`-Container und der Lis-

tencontainer `<ul>` selbst besitzen `id`-Attribute, um sie per CSS gezielt ansprechen zu können.

```
<div id="navigation">
  <ul id="hauptnavi">
    <li><a href="#" tabindex="1">Home</a></li>
    <li class="current">
      <a href="#" tabindex="2">Seite 1</a>
    </li>
    <li><a href="#" tabindex="3">Seite 2</a></li>
    <li><a href="#" tabindex="4">Seite 3</a></li>
  </ul>
</div>
```

Diese Ausgangskonfiguration wird links in Abbildung 7.15 gezeigt. Der Hyperlinktext des aktuellen Navigationspunkts wird über CSS bereits durch (beinahe) schwarze Textfarbe hervorgehoben, die anderen sind pauschal aufgehellt. Alle Linktexte sind zentriert und fett formatiert, der Unterstrich deaktiviert:

```
ul#hauptnavi  li  a { color: #999;                    font-
weight: bold;
                      text-align: center;
                      text-decoration: none;}
ul#hauptnavi  li.current  a { color: #555; }
```

Die restliche Designarbeit erfolgt vollständig auf CSS-Ebene. Zunächst werden die Aufzählungselemente der Liste per CSS-Anweisung entfernt.

```
ul#hauptnavi  { list-style-type:none; }
```

Das Ergebnis sehen Sie in Abbildung 7.15, rechts. Sehr spektakulär ist die Veränderung jedoch noch nicht.

Abbildung 7.15 Liste mit und ohne Bullets

Effektiver greift der folgende Schritt: Die Listenelemente werden nach links gefloatet (siehe Abbildung 7.16).

```
ul#hauptnavi  li  { float:left; }
```

Abbildung 7.16 Die Listenelemente sind gefloatet.

Um dem gewünschten Layout näher zu kommen, erhalten die Listenelemente nun eine Hintergrundfarbe. Ein Margin nach rechts wird hinzugefügt, um die einzelnen Navigationselemente ein wenig auseinanderzurücken:

```
ul#hauptnavi  li  { float:left;
                    background: #ddd;
                    margin-right:2px; }
```

Nun wird die Klasse `.current` eingesetzt, um den aktuellen Navigationspunkt von den anderen abzuheben. Hierfür bekommt dieser einen schwarzen Rahmen und eine weiße Hintergrundfarbe zugewiesen (siehe Abbildung 7.17):

```
ul#hauptnavi  li.current { border:1px solid #000;
                           background: #fff; }
```

Abbildung 7.17 Der Rahmen um das aktuelle Element

Um der Navigation etwas mehr Raum zu geben, werden den Hyperlinkcontainern eine Breite und Paddingwerte zugewiesen. Damit diese Anweisung greift, müssen auch diese Container nach links gefloatet werden. Zusätzlich wird die Schriftgröße etwas verringert:

```
ul#hauptnavi  li  a { float:left;
                      width:6em;
                      font-size:0.8em;
                      padding: 0.3em 1em 0.2em 1em;
                      color: #999;
                      font-weight: bold;
```

```
                    text-align: center;
                    text-decoration: none; }
```

Auch die Listenelemente bekommen nun eine Breite und Höhe in *em* zugewiesen:

```
ul#hauptnavi  li  { float:left;
                    background: #ddd;
                    margin-right:2px;
                    height:1.4em;
                    width:6.4em; }
```

Der Listencontainer `<ul>` erhält ein kleineres Padding nach links, um die Navigation optisch näher am Rand zu positionieren. Gleichzeitig wird er durch relative Positionierung etwas nach unten geschoben (Abbildung 7.18).

```
ul#hauptnavi { list-style-type:none;
               position:relative;
               top:0.15em;
               padding-left:0.5em; }
```

Abbildung 7.18 Die Navigation ist beinahe fertig.

Das Ergebnis ist beinahe schon wie gewünscht, erfordert aber noch etwas Nacharbeit. Der Navigationscontainer erhält eine Höhe und wird ebenfalls nach links gefloatet:

```
div#navigation  { float:left;
                  background:#fff;
                  height:2em;
                  padding:0; }
```

Last, but not least wird nun auch die Navigationsliste selbst gefloatet und erhält definierte Top- und Bottom-Margins:

```
ul#hauptnavi { float: left;
               list-style-type:none;
               position:relative;
               top:0.15em;
               padding-left:0.5em;
```

```
                margin-top:5px;
                margin-bottom:0px; }
```

Damit der untere schwarze Rand des aktuellen Listenelements verdeckt wird, bekommt der aktuelle Hyperlinkcontainer ebenfalls eine Höhe, eine weiße Hintergrundfarbe und einen weißen Bottom-Rahmen zugewiesen:

```
ul#hauptnavi  li.current a { color: #555;
                             border-bottom:1px solid #fff;
                             background:#fff;
                             height:1.4em; }
```

Abbildung 7.19 Das Ergebnis: Einfache listenbasierte Horizontalnavigation

7.7 Karteireitermenü auf Listenbasis

Optisch potentiell ansprechender als die rein CSS-basierten horizontalen Karteireiter sind solche mit Hintergrundbildern. Auf diesem Weg kann man beispielsweise eine (aktuell zumindest) beliebte Darstellung mit runden Ecken und Transparenzreflexen erzielen.

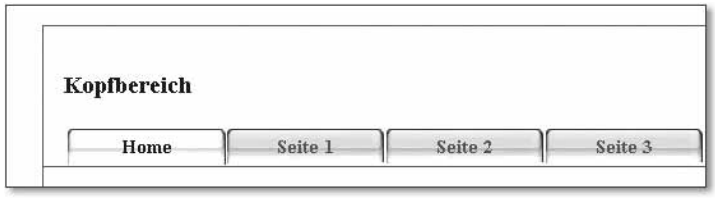

Abbildung 7.20 Die horizontale Karteireiternavigation

Die einfachste Variante besteht damit, dem Listenelement die gewünschte *Hintergrundgrafik* zuzuweisen und es dabei zu belassen. Dies funktioniert, solange die Schriftgröße des Linktextes dem ursprünglichen Entwurf entspricht und nicht skaliert wird. Wird die Schrift jedoch vergrößert (die meisten Browser lassen dies zu), so wird das exakt bemessene Hintergrundbild plötzlich zu klein. Ein Layout, das die *Skalierbarkeit* der Schrift mit einbezieht, muss notwendigerweise etwas aufwendiger sein, ist aber nicht unendlich kompliziert.

Die Quelltextgrundlage ist exakt dieselbe wie im vorigen Beispiel, auch eine `.current`-Klasse wurde wieder vergeben (in diesem Fall bindet sie ein anderes Hintergrundbild ein).

```
<div id="navigation">
  <ul id="hauptnavi">
    <li class="current">
      <a href="#" tabindex="1">Home</a>
    </li>
    <li><a href="#" tabindex="2">Seite 1</a></li>
    <li><a href="#" tabindex="3">Seite 2</a></li>
    <li><a href="#" tabindex="4">Seite 3</a></li>
  </ul>
</div>
```

Der Trick, der dem skalierbaren Karteireitermenü zugrunde liegt besteht darin, dass zwei Grafiken verwendet werden, die beide überdimensioniert sind (Abbildung 7.21). Die eine wird wie gehabt dem *Listencontainer* als Hintergrund zugewiesen und an dessen linker oberer Ecke angelegt. Es ist von diesem Bild (hier `tab_01.gif`) also nur soviel zu sehen, wie die Fläche des Containers freigibt. An der rechten Seite entsteht dadurch ein hässlicher Abriss in Form einer scharfen Kante.

An dieser Stelle springt die zweite beteiligte Grafik (`tab_02.gif`) ein, die nur die Aufgabe hat, diese rechte Kante zu überdecken. Sie wird dem *Ankercontainer* zugewiesen und an dessen rechter, oberer Ecke positioniert. Da der Ankercontainer im Inneren des Listenelements steht, verdeckt er dessen Hintergrundbild automatisch.

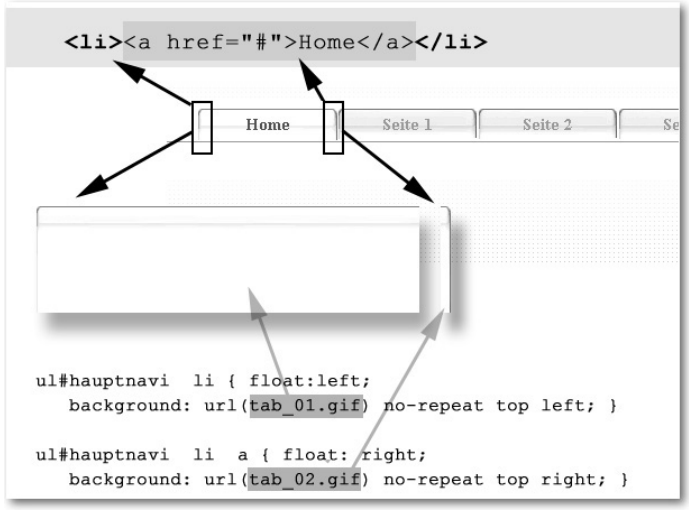

Abbildung 7.21 Die zwei Grafiken für skalierbare Karteireiternavigationen

Mit folgenden CSS-Anweisungen erhalten Sie eine horizontale Tab-Navigation, die eine Skalierbarkeit der Schrift ermöglicht. Es handelt sich lediglich um eine Erweiterung der zuvor erläuterten horizontalen Navigation, wobei im Wesentlichen die Hintergrundgrafiken hinzugefügt wurden. Allerdings ist diese Navigation zur Abwechslung komplett nach rechts gefloatet, wie in Abbildung 7.22 zu sehen ist:

```
ul#hauptnavi { float: right;
            position:relative;
            top:0.1em;
            margin-top:5px;
            margin-bottom:0px;
            list-style-type:none;
            width:27em; }

ul#hauptnavi  li {
            float: left;
            background: url(tab_01.gif) no-repeat top left;
            height:1.4em;
            width:6.4em;
            margin-right:2px; }

ul#hauptnavi  li.current {
            border-bottom:1px solid #fff;
            background: url(tab_01c.gif) no-repeat top left;
            height:1.45em; }

ul#hauptnavi  li  a {
            float: right;
            color: #999;
            height:1.2em;
            width:6em;
            font-size:0.8em;
            font-weight: bold;
            background: url(tab_02.gif) no-repeat top right;
            padding: 0.3em 1em 0.2em 1em;
            text-align: center;
            text-decoration: none; }

ul#hauptnavi  li.current a{
         color: #555;
         border-bottom:1px solid #fff;
         background: url(tab_02c.gif) no-repeat top right;
         height:1.35em; }
```

```
ul#hauptnavi  li  a:hover {
            color: #000; }

ul#hauptnavi  li.current {
            border-bottom:1px solid #fff; }
```

Abbildung 7.22 Die Navigation verträgt problemlos eine Schriftvergrößerung.

7.8 Grafikgestützte Navigation auf Listenbasis

Wenn für Navigationsmenüs Grafiken eingesetzt werden müssen, dann deshalb, weil das geforderte Layout anders nicht erreicht werden kann. Ein möglicher Grund besteht darin, dass die einzusetzende Schrift nicht auf allen Browsern (oder überhaupt nicht) verfügbar ist oder mit Effekten versehen werden muss, die mittels rein textbasierter Menüs nicht erreichbar sind.

Grundsätzlich gelten grafische Menüs als nicht barrierefrei – ein Manko, das jedoch, wie gleich gezeigt wird, umgangen werden kann. Ein definitiver Nachteil besteht in der etwas größeren Ladezeit durch die erforderlichen Grafiken und den höheren Aufwand an CSS, um einzelne Hintergrundgrafiken über ID an die Hyperlinkcontainer zu binden. Eine echte Barrierefreiheit lässt sich mit ein paar Tricks jedoch leicht erreichen.

Lässt man den Linktext einfach im Listenelement stehen, so verdeckt er die Grafik, was in diesem Fall nicht erwünscht ist. Den Text einfach wegzulassen (oder

durch geschützte Leerzeichen zu ersetzen), ist jedoch keine Lösung. Ein Linktext muss bleiben, damit Textbrowser oder Screenreader ihn auswerten können – ansonsten könnte man sich darauf beschränken, die Abmessungen der Container den Bedürfnissen der Hintergrundbilder anzupassen.

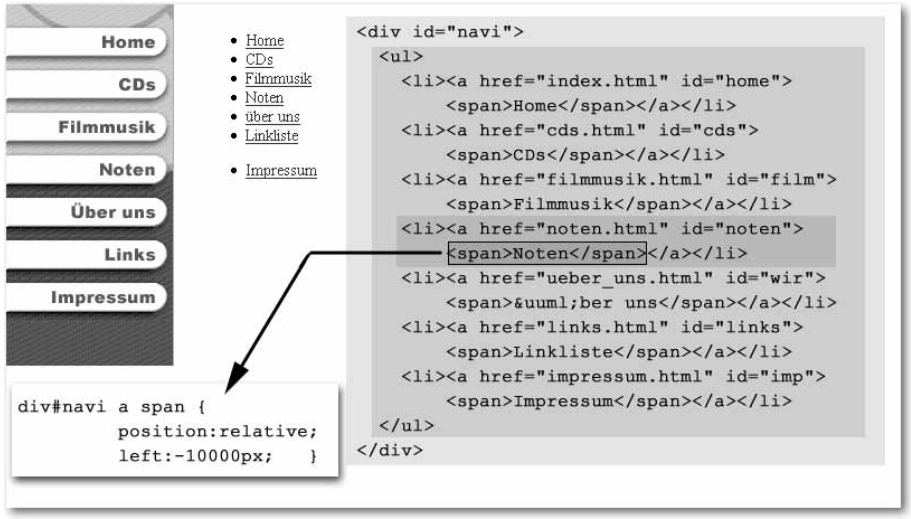

Abbildung 7.23 Das barrierefreie Grafikmenü

Die Festlegung dieser Abmessungen muss dennoch die erste Maßnahme sein:

```
div#navi  a  { display:block;
               width:165px; height:38px;
               background-repeat:no-repeat; }
```

Dies legt für die Hyperlinkcontainer im Navigationscontainer zum einen die Blockeigenschaft fest, die benötigt wird, damit die gesamte Containerfläche aktiviert wird. Zum anderen werden die Abmessungen so gesetzt, dass die gewünschten Hintergrundbilder die Fläche hinterlegen können. Da grundsätzlich für alle Navigationslinks gelten wird, dass das Hintergrundbild nicht wiederholt wird, kann auch dies gleich pauschal vereinbart werden.

Die XHTML-Quelltextstruktur ist folgende (beachten Sie das `id`-Attribut des `<a>`-Containers und den `<span>`-Tag um den eigentlichen Linktext):

```
<div id="navi">
  <ul>
    <li>
       <a href="index.html" id="home">
           <span>Home</span>
```

```
        </a>
    </li>
    ...
</ul>
</div>
```

Über das `id`-Attribut werden jedem Navigationslink nun individuell Hintergrundgrafiken zugewiesen, jeweils für den Normal- und den Hover-Zustand.

```
a#home       { background:url('../img/nav_home.gif'); }
a#home:hover { background:url('../img/nav_home-o.gif'); }
```

Sobald das Stylesheet abgeschaltet ist oder nicht ausgewertet werden kann, stehen jedoch die Grafiken und ihre visuelle Information nicht mehr zur Verfügung. An dieser Stelle sollen die Textinhalte »einspringen«, ansonsten müssen sie aus dem Weg geschafft werden, um die Grafiken nicht zu verdecken. Dies geschieht durch relative Positionierung der `<span>`-Container weit links außerhalb des Sichtbereiches:

```
div#navi a span { position:relative;
                  left:-10000px; }
```

Die Vorgehensweise entspricht also derjenigen beim Verstecken von Sprunglinks. Das Ergebnis ist dasselbe: Für *Screenreader* bleibt die Textinformation zugänglich, für reine *Textbrowser* (und Suchmaschinen) sieht die Navigation wie eine normale Liste aus – sieht man einmal von dem zusätzlichen `<span>`-Container ab. Alle anderen Nutzer erhalten ein voll funktionsfähiges grafisches Menü, das, ganz nebenbei den Vorzug der Barrierefreiheit bietet.

7.9 Zusammenfassung

Was Sie aus diesem Kapitel mitnehmen sollten:

- Mit einer gut entworfenen Navigation steht und fällt der Erfolg einer Website. Dies ist gleichzeitig ein Grundsatz der Barrierefreiheit.
- Die Funktionen der Navigation werden von verschiedenen Untergruppen übernommen, die als Haupt-, Bereichs-, Service- und Spontannavigation bezeichnet werden. Üblich sind auch die Begriffe »hierarchische Navigation«, »lokale Navigation«, »globale Navigation« und »Ad Hoc-Navigation«.

- Es gibt quelltextseitig viele Möglichkeiten des Markups von Navigationen, von denen die Auszeichnung als ungeordnete Liste die semantisch treffendste ist.
- Listenbasierte Navigationen können per CSS beliebig präsentiert werden, was auch die Darstellung als horizontale Navigation umfasst.

Wie geht es weiter?

Die Beschreibung der Grundlagen von HTML, CSS und des Aufbaus und Layouts von Webseiten ist mit diesem Kapitel abgeschlossen. Das folgende Kapitel 8, *Code-Aktualisierung nach XHTML,* erläutert, wie der Quellcode älterer Webseiten überarbeitet, entrümpelt und aktualisiert werden kann. Sehr gelegen kommt zu diesem Zweck das Tool HTML Tidy. Dennoch ist immer noch Handarbeit nötig, die jedoch bei geschickter Vorarbeit minimiert werden kann.

Häufig müssen alte oder schlecht geschriebene HTML-Dokumente überarbeitet werden, um sie beispielsweise in XHTML zu überführen. Tools wie HTML Tidy helfen bei der Aufgabe. In vielen Fällen muss nur noch geringfügig von Hand nachgearbeitet werden.

8 Code-Aktualisierung nach XHTML

Das Problem, alte oder fehlerhafte HTML-Dokumente in zeitgemäße, korrekte Codeform zu bringen, stellt sich jedem Webdesigner früher oder später. Dies kann bedeuten, eigene Werke von früher aktualisieren zu müssen (und wer hätte nicht seinerzeit mit Präsentationselementen gearbeitet!) oder Dokumente mit teilweise obskurer Herkunft zu überarbeiten – in diese Kategorie fallen auch die notorischen, mit Microsoft Word erzeugten »HTML-Seiten«.

Ein Umbau vorhandener Seiten macht Arbeit und verursacht daher Kosten. Das ist erstmal schlecht. Auf der anderen Seite steht jedoch auch (was gegenüber einem Auftraggeber ein gutes Argument darstellt) ein erheblicher Gewinn. Teilweise schlägt sich dies direkt in finanzieller Hinsicht nieder, in anderer Hinsicht ist auch eine verbesserte Zukunftsfähigkeit und das vermehrte Prestige durch Beachtung von Richtlinien und des »State Of The Art« des Webdesigns als positiv zu werten.

- Gewinn an Flexibilität für spätere Redesigns
- Gewinn an Barrierefreiheit
- Gewinn an Anpassungsmöglichkeiten für wechselnde User Agents
- Geringere Transferkosten durch verringerte Datenmenge
- Bessere Datenaustauschmöglichkeiten durch Einsatz von XHTML

8.1 Codeaktualisierung mit HTML Tidy

Praktisch ist es, wenn Codeaktualisierungen mit einem Programm erledigt werden können, das ähnlich wie ein HTML-Parser Codeergänzungen und -korrekturen automatisch vornimmt und das Ergebnis abspeichern kann. Umso besser, wenn auch im gleichen Zug überflüssiger Code entfernt und alles inklusive hinzugefügter Doctype-Deklaration und Namensraum in XHTML-Form gebracht

wird. Genau diese Aufgabe erledigt das von Dave Raggett initiierte **HTML Tidy**, das mittlerweile als Open-Source-Projekt weitergeführt wird.

Die Aufgabe von HTML Tidy besteht (wie der Name des Programms bereits aussagt) darin, HTML-Quellcode zu überarbeiten und zu säubern. Dies umfasst eine Überprüfung auf Korrektheit und die Korrektur von Fehlern, mindestens aber deren Anzeige in Form von Fehlermeldungen. Gleichzeitig kann der Quellcode »aufgeräumt« werden, indem überflüssige Elemente entfernt, fehlende ergänzt und die Lesbarkeit durch Einrückungen verbessert wird.

HTML Tidy kann beliebig HTML oder XHTML ausgeben und nach Wunsch Doctype-Deklarationen oder auch einen XML-Prolog hinzufügen. Die Ergebnisdatei ist abspeicherbar, wobei entweder die Quelldatei überschrieben oder eine neue Datei erzeugt wird. Per Stapelverarbeitung kann HTML Tidy ganze Verzeichnisse überprüfen.

Drei Mindestanforderungen der im Rahmen einer anstehenden Aktualisierung vorzunehmenden Arbeiten lassen sich recht gut automatisieren und mit HTML Tidy vornehmen.

1. **Entfernung aller Präsentationselemente**
   Überkommenes Markup wie `<font>` oder `<center>` und weitere Präsentationselemente und -attribute sollten entfernt werden.
2. **Umwandlung in korrektes XHTML**
   Dies betrifft Kleinschreibung der Element- und Attributnamen, korrekte Verschachtelung der Elemente sowie Hinzufügung des XHTML-Namensraums und der Doctype-Deklaration.
3. **Verbesserung der äußeren Form des Quellcodes**
   Hiermit sind Einrückungen im Quellcode gemeint, aber auch die Vereinheitlichung der Zeilenumbruchbefehle (CR und LF) oder eine Festlegung der maximalen Zeilenlänge.

8.1.1 HTML Tidy als Kommandozeilen-Tool

HTML Tidy ist von Haus aus ein einfaches **Kommandozeilen-Tool**, das für alle gängigen Betriebssysteme zur Verfügung steht. Sie erhalten die aktuelle Version des Tools bei Sourceforge.

- **HTML Tidy: http://tidy.sourceforge.net**

Unter Linux muss die Archivdatei von Tidy in das Verzeichnis `/usr/local/bin/` entpackt werden. Unter Windows reicht es aus, `tidy.exe` aus dem Zip zu entpacken und im Verzeichnis `C:\windows` oder `C:\winnt` abzulegen (oder je nach Windows-Version ein entsprechendes Verzeichnis).

Installation und Einsatz des Kommandozeilen-Tools

Für Windows existiert ein aktueller Installer `tidy-060405-setup.exe`, der Tidy sehr zuverlässig automatisch installiert und anmeldet. Dies hat den Vorteil, dass Sie sich nicht um einen Eintrag in den Klassenpfad zu kümmern brauchen. Sie starten HTML Tidy in einem geöffneten DOS-Fenster. Geben Sie dort `tidy -h` oder `tidy -?` ein, so listet Tidy Ihnen alle zur Verfügung stehenden Optionen auf (siehe Abbildung 8.1).

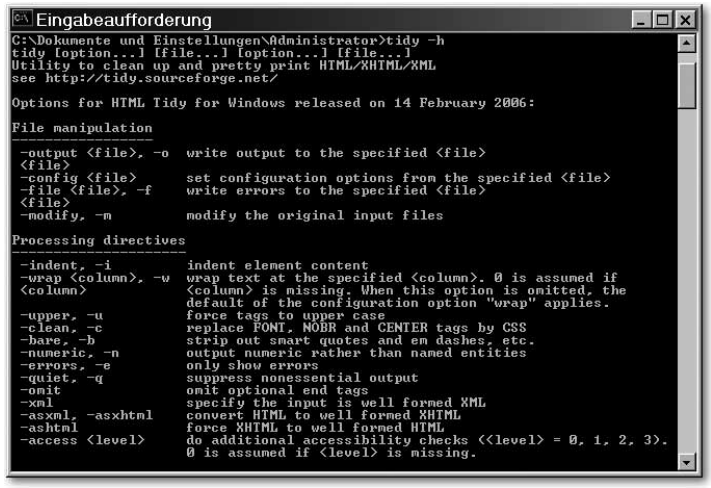

Abbildung 8.1 HTML Tidy im DOS-Fenster

Für die Kommandozeile stehen eine Reihe von Parametern zur Verfügung. Die wichtigsten sind in der folgenden Tabelle aufgelistet.

Parameter	Bedeutung
`-?`, `-h`, `-help`	"Help"; Tidy Hilfe; Auflistung aller Parameter
`-v`, `-version`	Gibt die Versionsnummer von Tidy aus
`-i`, `-indent`	"Indent"; Einrückungen der Elemente vornehmen
`-c`, `-clean`	"Clean"; Ersetzt <font> und durch CSS-Angaben
`-m`, `-modify`	"Modify"; Überschreibt Quelldatei mit Ergebnis
`-ashtml`	Gibt Ergebnis als wohlgeformtes HTML aus
`-asxml`	Gibt Ergebnis als wohlgeformtes XML bzw. XHTML aus
`-asxhtml`	Gibt Ergebnis als wohlgeformtes XHTML aus
`-u`, `-upper`	"Upper case"; Elementnamen in Großbuchstaben
`-o [datei]`, `-output [datei]`	"Output"; Gibt das Ergebnis in die genannte Datei aus

Tabelle 8.1 Aufrufparameter für HTML Tidy

Der Aufruf erfolgt über das Kommando `tidy`, gefolgt von einem oder mehreren Parametern. Gleichzeitig müssen Sie dem Programm den Pfad und Dateinamen des HTML-Dokuments nennen, das es verarbeiten soll. Das Prinzip ist wie folgt:

```
tidy [param] quelle
```

Dies kann konkret folgendermaßen geschehen (absoluter Pfad):

```
tidy -i c:\tidytest\beispiel.html
```

Wenn Sie sich die Pfadangabe ersparen wollen, wechseln Sie mit dem Prompt in den Ordner, in dem sich Ihre Quelldatei befindet. Sie brauchen anschließend nur noch den Dateinamen anzugeben (relativer Pfad):

```
tidy -i beispiel.html
```

Die Ausgabe erfolgt per Default ebenfalls im DOS-Fenster (Abbildung 8.2). Hier wurde das Programm mit `-i` lediglich angewiesen, Codeeinrückungen vorzunehmen. Das (sehr unvollständige) Quelldokument liegt für dieses Beispiel in einem Verzeichnis `tidytest` und hat folgenden Inhalt:

```
<title>Dies ist ein gueltiges HTML-Dokument</TITLE>
<P>Es enthaelt einen anklickbaren
<a HREF=anderes_dokument.html>Verweis</a>
auf ein anderes Dokument.
<br>Hier geht der Text noch etwas weiter. Das wars schon.
```

Listing 8.1 Die Datei »beispiel.html« für die Verarbeitung mit HTML Tidy

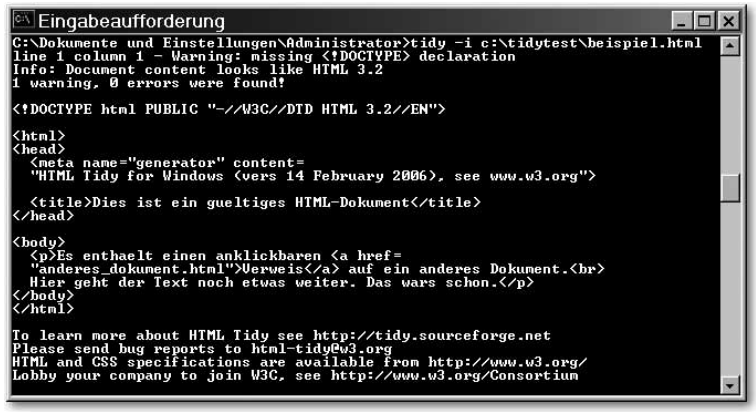

Abbildung 8.2 Die Ausgabe von Tidy im Kommandozeilenmodus

Wollen Sie das Ergebnis nicht im DOS-Fenster, sondern als Dateiausgabe erhalten, so können Sie Tidy ebenfalls einen Zielordner und einen Dateinamen nennen, der für die Ausgabe verwendet werden soll. Das Prinzip:

```
tidy [param] quelle > ziel
```

Alternativ können Sie auch schreiben:

```
tidy [param] quelle -o ziel
```

Tidy erzeugt am angegebenen Ort eine Datei des gewünschten Namens. Hier wird eine Datei `ergebnis.html` im Ordner des Quelldokuments erzeugt:

```
tidy -i c:\tidytest\beispiel.html > c:\tidytest\ergebnis.html
```

Die Ausgabe wurde wegen des Parameters `-i` mit Einrückungen formatiert. Tidy ergänzt die ausgelassenen Marken und fügt dem Dokument eine Doctype-Deklaration hinzu. Da in diesem Fall keine weiteren Angaben gemacht wurden, orientiert Tidy sich an Form und Inhalt des Dokuments, um einen passenden Typ zu finden. Das Quelldokument entspricht dem Standard von HTML 3.2, weshalb dies als Dokumenttyp gewählt wurde. Das Ergebnis hat folgende Form:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
  <meta name="generator" content="HTML Tidy for Windows
   (vers 14 February 2006), see www.w3.org">
  <title>Dies ist ein gueltiges HTML-Dokument</title>
</head>
<body>
  <p>Es enthaelt einen anklickbaren
  <a href="anderes_dokument.html">Verweis</a>
   auf ein anderes Dokument.<br>
   Hier geht der Text noch etwas weiter. Das wars schon.</p>
</body>
</html>
```

Listing 8.2 Die von HTML Tidy erzeugte Ausgabedatei »ergebnis.html«

Interessanter ist es natürlich, anstelle eines HTML-Dokuments ein XHTML-Dokument zu erzeugen. Hierfür braucht nur zusätzlich der Parameter `-asxhtml` übergeben zu werden. Um das vorige Ergebnis nicht zu überschreiben, wurde hier ein anderer Dateiname für die Ausgabe gewählt:

```
tidy -i -asxhtml c:\tidytest\beispiel.html > c:\tidytest\ergebnis_
xhtml.html
```

Das Ergebnis präsentiert sich nun als **XHTML »strict«**, mit entsprechender Doctype-Deklaration, XHTML-Namensraum und korrekter Syntax (achten Sie auf den intern abgeschlossenen Tag `<br />`).

```
<!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>
  <meta name="generator" content="HTML Tidy for Windows
   (vers 14 February 2006), see www.w3.org" />
  <title>Dies ist ein gueltiges HTML-Dokument</title>
</head>
<body>
  <p>Es enthaelt einen anklickbaren
  <a href="anderes_dokument.html">Verweis</a>
   auf ein anderes Dokument.<br />
   Hier geht der Text noch etwas weiter. Das wars schon.</p>
</body>
</html>
```

Listing 8.3 Die von HTML Tidy erzeugte Ausgabedatei »ergebnis_xhtml.html«

Wollen Sie, anstatt eine neue Datei zu erzeugen, direkt die Originaldatei mit der durch Tidy korrigierten Version überschreiben, so geben Sie den Parameter `-m` an. Beachten Sie aber, dass Sie im Vorfeld ein Backup der Datei(en) vornehmen sollten, da Tidy nicht die Möglichkeit einer Aktionsrücknahme bietet.

```
tidy -i -m -asxml c:\tidytest\beispiel.html
```

Die »Clean«-Option von Tidy ersetzt `<center>`- und `<font>`-Elemente durch `<div>`- und `<span>`-Container mit CSS-Klassen. Hierzu soll das Programm folgende Beispieldatei verarbeiten (beachten Sie, dass der Umlaut ä im Quelltext verwendet wurde):

```
<title>Dies ist ein HTML-Dokument mit Fehlern</TITLE>
<center>
<P>Es enthält einen
<font color="#ff0000">anklickbaren</font>
<a HREF=anderes_dokument.html>Verweis</a>
auf ein anderes Dokument.
<br>Hier geht der Text noch etwas <b>weiter</b>. Das wars schon.
</center>
```

Listing 8.4 Beispieldokument mit Präsentationsmarkup

Der Aufruf muss den Parameter `-c` beinhalten:

```
tidy -i -m -c -asxml c:\tidytest\font_test.html
```

Im Ergebnisdokument wird ein `<style>`-Container eingefügt. Tidy generiert die entsprechenden CSS-Klassennamen und weist diese den `<div>`- und `<span>`-Containern zu, die `<center>` und `<font>` ersetzen. Anstelle des Umlauts ä wurde das Entity `ä` eingesetzt. Beachten Sie, dass diesmal der »transitional«-Doctype gewählt wurde, da dieser dem Ausgangsdokument verwandter ist:

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta name="generator" content="HTML Tidy for Windows
  (vers 14 February 2006), see www.w3.org" />
  <title>Dies ist ein HTML-Dokument mit Fehlern</title>
  <style type="text/css">
    /*<![CDATA[*/
     div.c2 {text-align: center}
     span.c1 {color: #FF0000}
   /*]]>*/
  </style>
</head>
<body>
  <div class="c2">
    <p>Es enth&auml;lt einen
    <span class="c1">anklickbaren</span>
    <a href="anderes_dokument.html">Verweis</a>
     auf ein anderes Dokument.<br />
     Hier geht der Text noch etwas <b>weiter</b>.
     Das wars schon.</p>
  </div>
</body>
</html>
```

Weitere Optionsparameter für HTML Tidy

Es stehen neben den bereits bekannten Parametern noch eine Vielzahl weiterer Optionen zur Verfügung, von denen die wichtigsten hier vorgestellt werden sollen. Eine vollständige Übersicht erhalten Sie mittels der Eingabe

```
tidy -help-config
```

Wollen Sie eine dieser fortgeschritteneren Optionen an der Kommandozeile direkt einsetzen (oder ausprobieren), so wählen Sie hierfür die Form

```
--[option_1] [wert] --[option_2] [wert] ...
```

also beispielsweise zum Erzwingen einer XML-Deklaration:

```
tidy --add-xml-decl yes beispiel.html
```

Insgesamt stehen rund siebzig derartige Optionen zur Verfügung, von denen hier deshalb nur die wichtigsten vorgestellt werden sollen.

Option	Default	Werte
write-back	no	yes, no

Legt fest, ob die Quelldatei durch das Verarbeitungsergebnis überschrieben werden soll, was der Übergabe des Parameters `-m` oder `-modify` entspricht.

[!] *Achtung*: Gefährlich in Zusammenhang mit Stapelverarbeitungen.

Option	Default	Werte
drop-font-tags	no	yes, no

Legt fest, ob die Elemente `<font>` und `<center>` verworfen werden, ohne dass deren Wirkung mit CSS-Regeln nachgebildet wird. Die Option `drop-font-tags` ist unabhängig von der Option `clean`.

Option	Default	Werte
drop-empty-paras	yes	yes, no

Legt fest, ob Tidy leere Textabsatzcontainer `<p></p>` verwerfen soll.

Option	Default	Werte
enclose-text	no	yes, no

Legt fest, ob Tidy unmittelbar im `<body>`-Element vorgefundenen Text mit `<p>`-Containern umgeben soll.

[»] *Hinweis:* Text ist in `<body>` in HTML gestattet, nicht jedoch in XHTML. Ohne umgebenden Container ist Text für CSS nicht korrekt erfassbar.

Option	Default	Werte
logical-emphasis	no	yes, no

Mit dieser Option kann Tidy angewiesen werden, sämtliche im Dokument erscheinenden Elemente <i> und <b> durch ihre logischen Pendants und zu ersetzen.

Option	Default	Werte
lower-literals	yes	yes, no

In XHTML müssen vordefinierte, für ein Attribut zur Auswahl stehende Werte stets klein geschrieben werden (z. B. »left«, »right«, »center«). Diese Option sorgt im Bedarfsfall für deren Umwandlung in Kleinbuchstaben.

Option	Default	Werte
output-xhtml	no	yes, no

Legt fest, ob Tidy eine Ergebnisdatei in XHTML-Form erzeugen soll. In diesem Fall setzt Tidy den XHTML-Namensraum und eine Doctype-Deklaration dazu. Ist eine bestimmte Doctype-Version festgelegt, so versucht Tidy, das Dokument entsprechend zu korrigieren.

Option	Default	Werte
quote-ampersand	yes	yes, no

Legt fest, ob Tidy im Textinhalt erscheinende einzeln stehende Ampersand-Zeichen durch das Entity & ersetzen soll.

Option	Default	Werte
quote-marks	no	yes, no

Ersetzt doppelte Anführungszeichen im Text durch das Entity " und einfache Anführungszeichen durch das Character-Entity ' (das hier eigentlich korrektere Entity ' gilt als nur mangelhaft unterstützt).

Option	Default	Werte
add-xml-decl	no	yes, no

Legt fest, ob Tidy bei der Ausgabe im Format XML oder XHTML eine XML-Deklaration der Form <?xml version="1.0"?> voranstellen soll. Die Anweisung wird jedoch ignoriert, wenn das Eingabedokument bereits eine XML-Deklaration besitzt.

Ist die `char-encoding`-Option gesetzt, wird der XML-Deklaration ein `encoding`-Attribut mit dem dort bestimmten Wert hinzugefügt. Ist `char-encoding` auf einen anderen Wert als `"ascii"` gesetzt, so wird einer XML- oder XHTML-Ausgabe stets eine XML-Deklaration vorangesetzt, ungeachtet der Festlegung in der Option `add-xml-decl`.

Option	Default	Werte
`char-encoding`	ascii	raw, ascii, latin0, latin1, utf8, iso2022, mac, win1252, ibm858, utf16le, utf16be, utf16, big5, shiftjis

Diese Option bestimmt das Encoding-Schema, das von Tidy sowohl für das Eingabe- als auch für das Ausgabedokument verwendet werden soll. Ist das Dokument beispielsweise in ISO-8895–1 (»latin1«) encodiert und ist `char-encoding` auf `"ascii"` gesetzt, so ersetzt Tidy alle Zeichen oberhalb des Zeichencodes 127 durch Entities. Ist `char-encoding` dagegen auf `"latin1"` gesetzt, werden erst Zeichen oberhalb von 255 ersetzt.

Option	Default	Werte
`doctype`	auto	omit, auto, strict, loose, user

Die `doctype`-Option legt fest, welche Doctype-Deklaration Tidy für das Ergebnisdokument verwenden soll. Für die Defaulteinstellung `"auto"` versucht Tidy, die passende Deklaration aus dem Dokumentinhalt zu erschließen. Der Wert `"omit"` hingegen unterdrückt die Ausgabe einer Doctype-Deklaration. Die Werte `"strict"` und `"loose"` legen eine *strict*- respektive *transitional*-Deklaration fest. Alternativ kann über `"user"` ein eigener Doctype-Deklarationsstring festgelegt werden.

Option	Default	Werte
`accessibility-check`	0	0, 1, 2, 3

Tidy kann die Barrierefreiheit eines Dokuments gemäß der drei Prioritätsstufen der Web Content Accessibility Guidelines 1.0 testen. Hierfür muss für die Option `accessibility-check` der Wert "1" (Priority 1 Checks), "2" (Priority 2 Checks) oder "3" (Priority 3 Checks) angegeben werden. Für gefundene Fehler gemäß der eingestellten Stufe gibt Tidy einen Fehlercode aus.

Tidy-Fehlercodes für Accessibility-Checks

Eine detaillierte Erläuterung der WAI-Fehlermeldungen von Tidy erhalten Sie beim »Adaptive Technology Resource Centre« der Universtät von Toronto unter dem URL **www.aprompt.ca/Tidy/accessibilitychecks.html**.

Konfiguration von HTML Tidy

Wenn Sie nicht stets alle Parameter oder gar die höheren Optionen mit Werten an der Kommandozeile eingeben wollen, können Sie für HTML Tidy auch beliebige **Verarbeitungskonfigurationen** zusammenstellen und diese in einer Datei ablegen. Es handelt sich um einfache Textdateien, die etwa folgende Verarbeitungsbefehle enthalten können:

```
clean: yes
logical-emphasis: yes
drop-empty-paras: yes
enclose-text: yes
indent: yes
output-xhtml: yes
uppercase-tags: no
uppercase-attributes: no
```

Listing 8.5 Beispiel eine Konfigurationsdatei für HTML Tidy

Haben Sie eine Konfigurationsdatei erstellt, so können Sie sie für eine anstehende Verarbeitung laden. HTML Tidy sucht die Datei im aktuellen Verzeichnis:

```
tidy  -config tidyconf.txt  beispiel.html
```

Batch-Verarbeitung mit dem Kommandozeilen-Tool

Sie können auch relativ einfach eine Stapelverarbeitung eines kompletten Ordnerinhalts mit HTML Tidy vornehmen, wobei auch eine Konfigurationsdatei zur Steuerung der Verarbeitung einbezogen werden kann. Übergeben Sie den Parameter `-m`, so werden alle verarbeiteten HTML-Dokumente durch die Ausgabe von Tidy überschrieben. In diesem Fall ist es ratsam, vorher eine Sicherungskopie der zu verarbeitenden Dateien zu machen.

Öffnen Sie wie gehabt ein DOS-Fenster und wechseln Sie in das Verzeichnis, dessen Inhalt verarbeitet werden soll. Geben Sie den folgenden Befehl ein:

```
for %i in (*.html) do tidy -m "%i"
```

Falls Sie Dateien mit der Endung `.htm` oder anders verarbeiten wollen, passen Sie den Befehl entsprechend an. Dann drücken Sie ↵. Das war's schon. Mit der

Konfigurationsdatei arbeiten Sie entsprechend (wenn Sie `write-back: yes` in die Konfiguration eingefügt haben, kann der Parameter `-m` entfallen):

```
for %i in (*.html) do tidy -config tidyconfig.txt -m "%i"
```

Im Anschluss sollten Sie sich die verarbeiteten Dateien zumindest stichprobenweise ansehen, um sicherzugehen, dass alles nach Wunsch verlaufen ist.

8.2 GUI-Versionen von HTML Tidy

Es existieren auch Webeditoren, die HTML Tidy integrieren und so in einer komfortableren Oberfläche nutzbar machen (als Beispiele zu nennen wären hier **tsWebEditor**, **TopStyle Pro** oder **HTML-Kit** – für Adobe **Dreamweaver** ist eine Erweiterung verfügbar und auch für **Eclipse** existiert ein Plug-in). Unter Linux kann Tidy in die Editoren **Bluefish** und **Quanta** integriert werden, für MacOS existiert für BBEdit ein Plug-in »BBTidy«. Unter Windows ist HTML Tidy auch in zwei autarken Versionen mit grafischer Oberfläche erhältlich – **Tidy GUI** und **Tidy UI**, die nachfolgend beschrieben werden.

- **tsWebEditor: http://tswebeditor.net.tc/**
- **Top Style Pro: www.bradsoft.com**
- **HTML-Kit: www.chami.com/HTML-Kit/**
- **Bluefish: http://bluefish.openoffice.nl/**
- **Quanta: http://quanta.sourceforge.net/**
- **Eclipse Plugin HTML Tidy: http://eclipsetidy.sourceforge.net/**

8.2.1 Tidy UI für Windows

Bei **Tidy UI** von Charles Reitzel handelt es sich um eine Bedienoberfläche, die auf HTML Tidy aufsetzt. Sie können hier HTML-Dokumente öffnen oder direkt im Editorfenster von Tidy UI erstellen und durch Tidy verarbeiten lassen.

- **Tidy UI: http://users.rcn.com/creitzel/tidy.html**

Ein über **File > Open > HTML** geöffnetes Dokument wird durch Klick auf den Button **Tidy** verarbeitet und ist im Editorfenster mit dem Reiter »Tidy« sichtbar. Sie können per Klick zum Originaldokument wechseln und auch per Cut and Paste Codefragmente zwischen beiden Fenstern hin- und her kopieren oder die Ansichten von Hand editieren.

Abbildung 8.3 Die graphische Oberfläche von HTML Tidy: Tidy UI

Ein Klick auf den Button **Replace** lädt das Verarbeitungsergebnis in das Fenster »Original«. Von dort aus können Sie es über den **Save**-Button (Diskettensymbol) direkt speichern, wobei das Quelldokument überschrieben wird. Ausgehend vom »Tidy«-Fenster ist ebenfalls eine Speicherung möglich, jedoch nur über das Menü **File > Save as** (hierbei können Sie natürlich ebenfalls das Quelldokument überschreiben, wenn Sie es wünschen).

Die Parameter zur Konfiguration von Tidy UI sind gruppiert in der linken Fensterhälfte zugänglich, wobei per Karteireiter zwischen den Gruppen gewechselt wird. Einen Parameter setzen Sie, indem Sie die entsprechende Option in der Liste anklicken. Es öffnet sich jeweils ein Popup-Fenster, in dem Sie per Dropdown-Menü unter den möglichen Einstellungen wählen können. Der **Hilfe**-Button öffnet jeweils eine Online-Erläuterung zur Option im Browser.

Abbildung 8.4 Option zur Wahl des Zeilenendezeichens

Zugänglich sind sämtliche höheren Optionen, die HTML Tidy zur Verfügung stellt. Sicherlich interessant darunter ist die Option, die **Barrierefreiheit** nach WAI zu prüfen. Dies geschieht durch Anwahl des Parameters `accessibility-check` in der Palette **Warnings**.

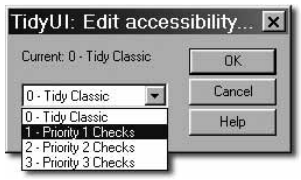

Abbildung 8.5 Auswahl des Checks nach WAI-Prioritäten

Die vorgenommenen Einstellungen bleiben erhalten. Alternativ können auch Konfigurationsdateien erstellt und wieder geladen werden. Hierzu dient ebenso wie zum Öffnen der zu testenden HTML-Dokumente das Menü **File**.

8.2.2 Tidy GUI für Windows

Die andere Oberfläche, **Tidy GUI** (Abbildung 8.6) von André Blavier wirkt anfänglich spartanischer, ist jedoch ebenso simpel zu bedienen und zu konfigurieren wie Tidy UI. Das Programm wird derzeit nicht weiterentwickelt, ist aber noch immer erhältlich. Unter der Webadresse steht auch eine deutschsprachige Version zum Download bereit.

- **Tidy GUI: http://perso.orange.fr/ablavier/TidyGUI/**

Um eine Datei zu verarbeiten, wählen Sie diese lediglich mit einem **Durchsuchen**-Dialog an und klicken auf den Button **Tidy!**.

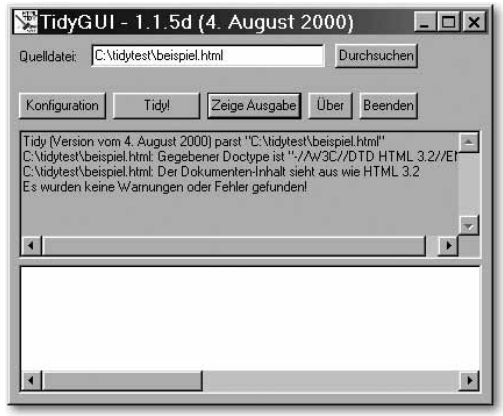

Abbildung 8.6 Tidy GUI mit Verarbeitungsmeldungen

Das Hauptfenster von Tidy GUI zeigt nur die Meldungen der erfolgten Verarbeitung, nicht aber die Ergebnisdatei. Um diese zu sehen, klicken Sie auf den Button **Zeige Ausgabe**, der ein weiteres Fenster mit dem Verarbeitungsergebnis öffnet (siehe Abbildung 8.7). In diesem Fenster können Sie auch nachfolgende Editierarbeiten vornehmen, wofür allerdings zunächst erst die Checkbox **Schreibgeschützt** deaktiviert werden muss. Danach können Sie das Ergebnis über den Dialog **Speichern als...** speichern.

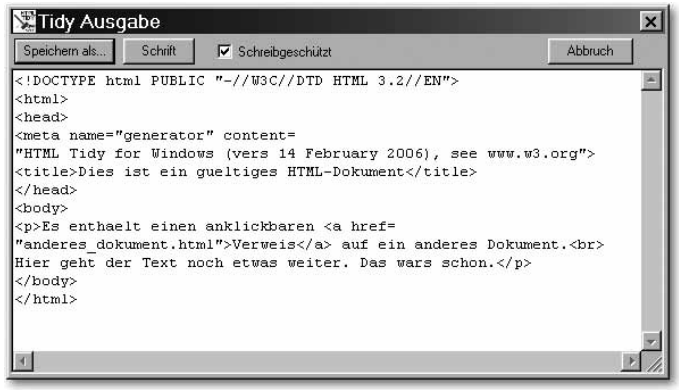

Abbildung 8.7 Ausgabe- und Editorfenster von Tidy GUI

Auch die Konfiguration von Tidy GUI findet in einem separaten Fenster statt, das Sie, ausgehend vom Hauptfenster über **Konfiguration** öffnen (siehe Abbildung 8.8). Die Möglichkeiten entsprechen denen von Tidy UI, sind aber übersichtlicher mittels einer Anzahl von Karteireitern zu erreichen. Eine Konfiguration kann mit **Speichern als...** gespeichert und später über **Laden** erneut geladen werden.

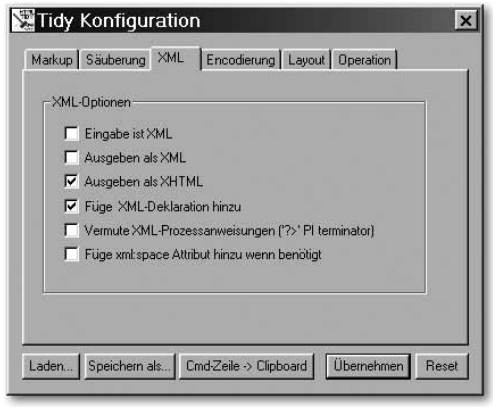

Abbildung 8.8 Konfiguration von Tidy GUI

8.2.3 HTML Tidy in HTML-Kit

In den (nebenbei bemerkt: höchst empfehlenswerten) Editor **HTML-Kit** ist Tidy bereits integriert und kann entweder über den Tastatur-Shortcut F9 oder per Icon-Button aufgerufen und auf das aktuell bearbeitete Dokument angewendet werden. Im »Splitscreen«-Modus können Sie gleichzeitig das Quelldokument und die durch Tidy überarbeitete Version nebeneinander betrachten (siehe Abbildung 8.9; links die Editor-Ansicht, rechts der Tidy-Output).

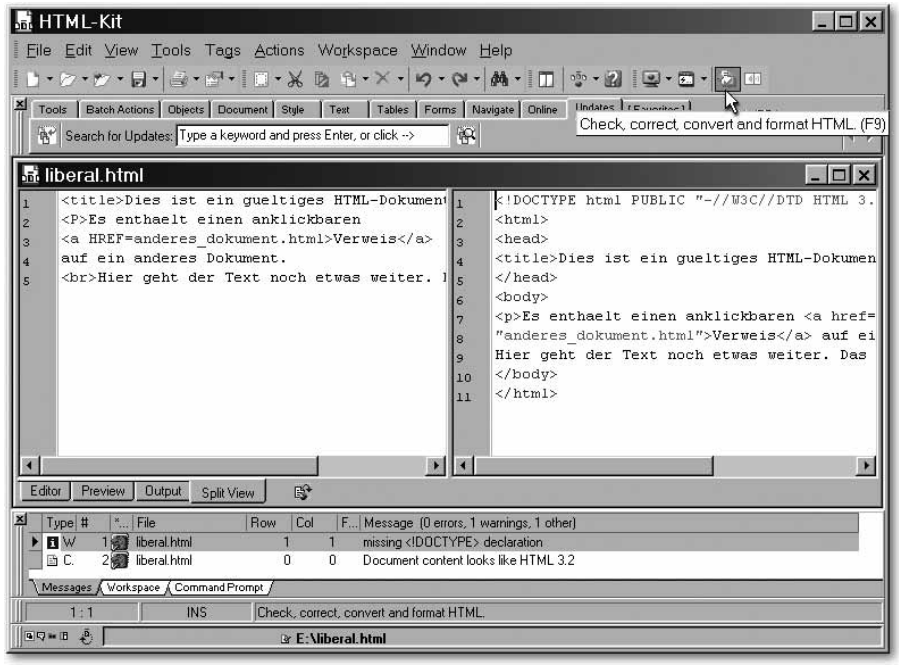

Abbildung 8.9 HTML Tidy in HTML-Kit integriert

Konfiguration von HTML Tidy in HTML-Kit

HTML-Kit setzt Tidy in der Ausgangskonfiguration des Tools ein, so dass nicht automatisch XHTML erzeugt wird. Will man beispielsweise XHTML »strict« mit XML-Deklaration und Encoding nach ISO-8859–1 erhalten, so muss man die Einstellungen entsprechend anpassen. Dies geschieht über das Menü **Tools > Check Code Using TIDY > Add New Config**. Dies entspricht Schritt ❶ in Abbildung 8.10. Der aktive Reiter ❷ muss »TIDY« sein. In dieser Palette können Sie sämtliche Tidy-Konfigurationen vornehmen. In diesem Fall soll die Ausgabe auf XHTML gesetzt werden ❸, das Encoding auf ISO-8859–1 ❹ und die DTD auf »strict« ❺. Das Häkchen für das Hinzufügen einer XML-Deklaration ist standardmäßig gesetzt.

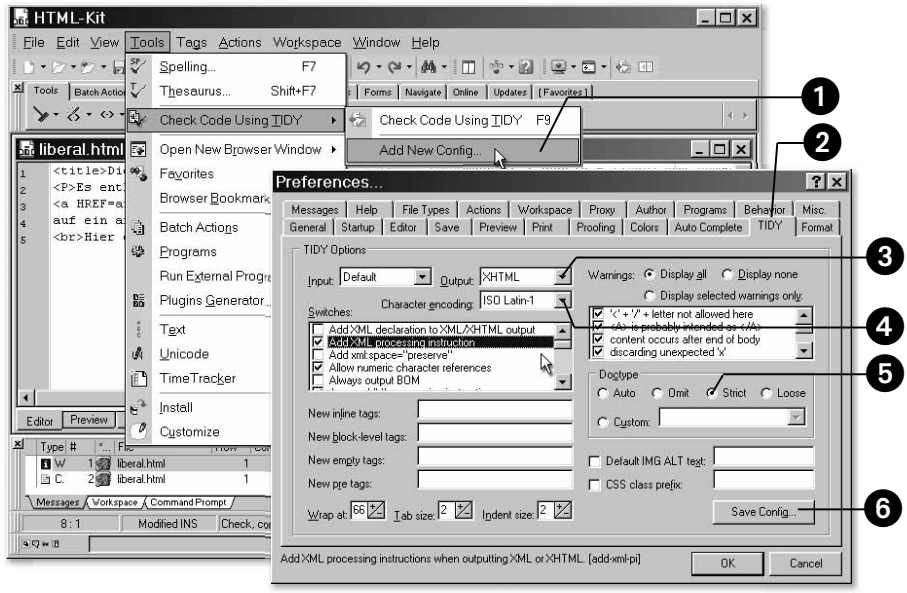

Abbildung 8.10 Erzeugen einer Tidy-Konfigurationsdatei in HTML-Kit

Die Konfiguration kann nun gespeichert werden ❻. Dies geschieht als `.htc`-Datei, die HTML-Kit im Ordner `data/default` ablegt. Im vorliegenden Fall wurde sie als `xhtml_strict_iso.htc` gesichert. Sie können beliebig viele unterschiedliche Ausgabekonfigurationen ablegen. Eine solche Konfiguration rufen Sie ebenfalls wieder über das Menü **Tools > Check Code Using TIDY** auf. Hier finden Sie für jede Konfiguration einen neuen Menüpunkt im Dropdown-Menü (siehe Abbildung 8.11).

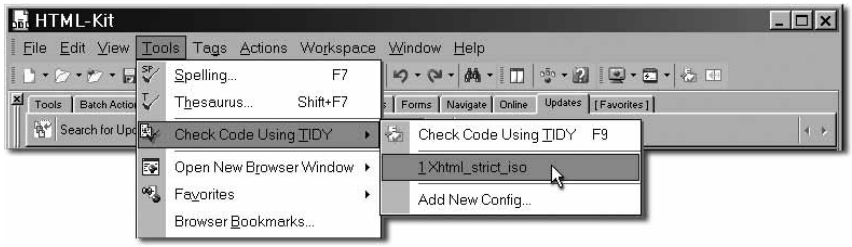

Abbildung 8.11 Auswahl einer gespeicherten Ausgabekonfiguration

Stapelverarbeitung durch HTML Tidy in HTML-Kit

Da die Prüfung von Einzeldateien mit Tidy mühselig ist, kommt es gelegen, dass HTML-Kit auch die Möglichkeit bietet, per Stapelverarbeitung eine beliebige Zahl

von HTML-Dokumenten »auf einen Rutsch« testen zu lassen. Die Dateien müssen in einem Verzeichnis liegen. Wählen Sie den Menüpunkt **Actions > Batch Actions > Check Code using HTML Tidy** (siehe Abbildung 8.12)

Der Batch-Modus in HTML-Kit dient nur zur Codeprüfung

Es findet definitiv nur eine Überprüfung der Syntax statt, keine automatische Verarbeitung und Codeanpassung. Letzteres ohne weitere Kontrolle vorzunehmen, wird vom Hersteller als zu riskant empfunden.

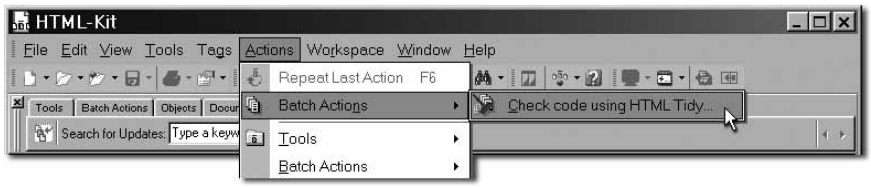

Abbildung 8.12 Auswahl der Stapelverarbeitung mit HTML Tidy in HTML-Kit

Der Ordner, dessen Inhalt verarbeitet werden soll, muss zunächst benannt werden. Dies geschieht leider etwas unkomfortabel, indem Sie den Pfad manuell in das Eingabefeld des sich nun öffnenden Fensters eintragen. Lassen Sie die Checkbox **Look in sub folders** aktiviert, so werden auch Dateien in Unterordnern des angegebenen Verzeichnisses verarbeitet. Sobald Sie auf den Button **Start** klicken, werden die Dateien geprüft und die je Datei gefundenen Fehler unter **Results** gezeigt.

Über die Option **Open** kann eine fehlerhafte Datei im Editor geöffnet werden. Sie können sie entweder von Hand editieren oder dies durch Tidy vornehmen lassen. Dies geschieht durch Klick auf das Tidy-Icon oder über das Menü **Actions > Tools > HTML Tidy > Invoke HTML Tidy**. Lassen Sie sich das Ergebnis in Splitview anzeigen und nehmen Sie gegebenenfalls weitere Änderungen von Hand vor.

Sobald Sie zufrieden sind, kopieren Sie die Voransicht (diese können Sie nicht direkt speichern) über **Edit > Copy Output To Editor** in das Editorfenster. Bei Speicherung wird nun die Originaldatei überschrieben. Speichern Sie das Ergebnis daher erst ab, wenn Sie sicher sind, dass alles korrekt ist – auch HTML-Kit legt kein Dateibackup an.

8.3 Codeaktualisierung von Hand

Grundsätzlich muss bei einer Aktualisierung einer bestehenden Seite der Webprogrammierer von Hand eingreifen, obwohl Werkzeuge wie Tidy gute unterstützende und vorbereitende Dienste leisten. HTML Tidy ist zwar in der Lage, den Quellcode mechanisch zu reorganisieren, scheitert aber naturgemäß an Aufgaben, für deren Erledigung die Beurteilung des Codes erforderlich ist. Von Hand müssen (mindestens) noch immer vorgenommen werden:

- **Beseitigung von Platzhaltergrafiken und anderem Beiwerk**
  Alle Grafiken, die zu Positionierungszwecken oder als Abstandshalter dienen (Stichwort »Single-Pixel-Gif-Trick« von David Siegel, bei dem unsichtbare, größenskalierte Grafiken als Positionierungshilfe verwendet werden[1]) sollten entfernt und ihre Wirkung durch CSS-Anweisungen nachempfunden werden. Dies ist auch dann sinnvoll, wenn die Grafiken ein Tabellenlayout stabilisieren, das beibehalten wird.
- **Entfernen von Tabellenlayout**
  Der Schritt von der Layouttabelle zum `<div>`-Layout lohnt sich, sobald sich das Webangebot an verschiedene Gattungen von User Agents richtet und entsprechend flexibel ausgegeben werden muss. Obendrein bedeutet er mehr Barrierefreiheit. Müssen aus was für Gründen auch immer die Layouttabellen bestehen bleiben, können sie allerdings ebenfalls mit CSS formatiert werden.
- **Setzen von `id`-Identifiern für Dokumentbereiche**
  Für Layoutzwecke müssen die einzelnen Dokumentbereiche mit `id`-Attributen versehen werden, um sie durch CSS-Anweisungen erfassen zu können und das gewünschte Layout zu ermöglichen. In der Regel wird man auf eine Struktur aus `<div>`-Containern zugreifen. Muss man eine Layouttabelle stylen, so bekommen die entsprechenden Datenzellen einen Identifier.
- **Setzen sinnvoller `alt`-Texte für Grafiken und andere Medien**
  Hier muss natürlich der Bildinhalt beschrieben werden. Tidy kann dies nicht. Nicht inhaltsrelevante Illustrationen erhalten einen leeren `alt`-Text.
- **Setzen der `title`-Attribute, wo erforderlich**
  Dies gilt (mindestens) für Hyperlinks, deren Ziel über `title` angegeben werden muss, kann aber auch andere Elemente betreffen.
- **Umbau von Formularen zur Verbesserung der Barrierefreiheit**
  Formulare sollten `<fieldset>` und `<label>`-Elemente erhalten. Zusätzlich kann über `tabindex`- oder `accesskey`-Attribute nachgedacht werden.

1 Auch aktuelle Imageslice-Tools setzen dies bei ausgegebenen HTML-Tabellen noch ein!

- **Semantisch korrektes Markup des Textinhalts**
  Elemente sollten in ihrer semantischen Bedeutung verwendet sein. Die Überprüfung der Überschriftenhierarchie und der Verwendung von strukturell bedeutsamen Tags zur Auszeichnung der Inhalte fällt in diesen Bereich. Das Einfügen von logischen Elementen wie `<abbr>` oder `<acronym>` innerhalb des Seiteninhaltes muss von Hand vorgenommen werden. Beseitigt werden muss auch Markup, das in Präsentationsabsicht missbraucht wurde – Tidy kann nicht entscheiden, ob `<blockquote>` zur Einrückung dient oder wirklich ein Zitat kennzeichnet. Ebenso ist die Verwendung des Konstrukts `<br> ` als Abstandhalter für Tidy nicht als Zweckentfremdung erkennbar.
- **Vergabe von Styleklassen**
  Tidy vergibt Styleklassen beim Ersatz von Präsentationsmarkup, was jedoch meist eine wenig brauchbare Grundlage darstellt. Besser ist die komplette Entfernung eines solchen Markups mit der Option `drop-font-tags` (anstelle von `clean`). Anschließend müssen jedoch Styleklassen von Hand vergeben werden.
- Ü**berprüfung von Farben und Kontrasten**
  Text und Hintergrund sollten ausreichend kontrastieren. (Gut, das hört sich trivial an. Man sollte aber nicht meinen, dass es immer beachtet würde!)

… und derlei Dinge mehr.

8.3.1 Praxisbeispiel eines 30 Minuten Redesigns

Im Prinzip haben Tabellenlayouts keine Daseinsberechtigung mehr, sind aber noch allerorten anzutreffen. Soll eine ältere Website überarbeitet werden, ist die Chance verhältnismäßig groß, auf ein derartiges Artefakt zu treffen. In Fällen, wo eine vollständige Überarbeitung unökonomisch oder aus anderen Gründen nicht opportun ist, kann man jedoch auch ein Tabellenlayout halbwegs vertretbar in die heutige Zeit hinüber retten (die Betonung liegt auf »halbwegs«).

Schritt 1 – Klärung des verfügbaren Budgets und des Zeitrahmens

Im vorliegenden Fall ging das Budget gegen Null (Einladung zum Essen), und der zur Verfügung stehende Zeitrahmen belief sich auf eine gute halbe Stunde (bis das Essen fertig war). Die Ausgangslage war ein stark angejahrtes Layout, das erstens auf Tabellen beruhte und zweitens mit Präsentationsmarkup (teilweise anstelle von Überschriftentags) und Platzhaltergrafiken durchsetzt war.

```
<td height="200" background="Bilder/Frame_
Oben200px.jpg">  <div align="center">
  <font size="5" color="white" face="Helvetica, sans-serif">
   Kursangebote
  <br><br>
  </font>
 </div>
</td>
```

Listing 8.6 Pseudoüberschrift in der alten Version (Kopfzeile der Tabelle)

Auch die Semantik ließ zu wünschen übrig – zum vermutlichen Zeitpunkt der Erstellung des Layouts war es jedoch noch nicht üblich, Navigationsmenüs listenbasiert zu gestalten (auch der Rest entspricht den Gepflogenheiten jener Zeit). Die Tabelle wies eine fixe Breite auf. Ebenso waren den Zeilen und Zellen Breiten, Höhen und Hintergrundgrafiken zugewiesen. Aus Zeitgründen wurde entschieden, das Tabellengerüst beizubehalten.

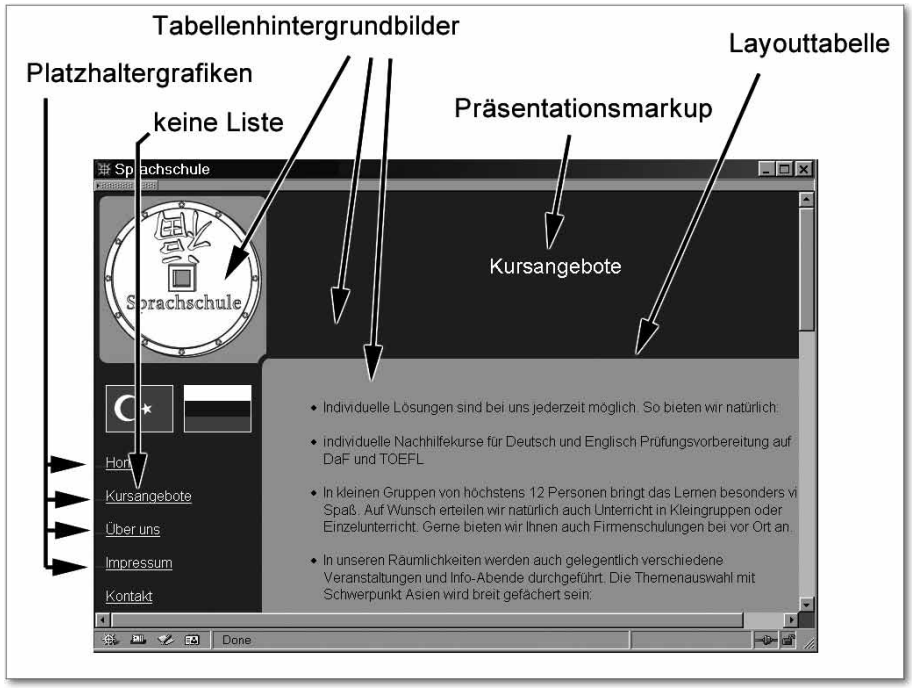

Abbildung 8.13 Höchst altmodisches Tabellenlayout als Ausgangslage

Schritt 2 – HTML Tidy

Zunächst wurde HTML Tidy auf den Quellcode losgelassen. Als Ziel-Dokumenttyp wurde XHTML »transitional« gewählt. Im ersten Schritt wird so zumindest (mit der Einstellung `drop-font-tags`) das Präsentationsmarkup beseitigt und die Quelltextqualität erhöht.

Die Kopfzelle stellt sich etwas besser nun folgendermaßen dar. Allerdings ist es noch immer keine Überschrift und der `<div>`-Container und der überflüssige Zeilenumbruch stören:

```
<td height="200" background="Bilder/Frame_Oben200px.jpg">
<div style="text-align: center">
 Kursangebote
</div><br />
</td>
```

Listing 8.7 Leider noch keine Überschrift – nach Durchlauf durch Tidy

Wenig verbessert zeigt sich der Navigationsbereich (den alten Zustand lasse ich außen vor). Hier stören nach wie vor die Platzhaltergrafiken. Die ungünstig gelöste Trennung der Navigationspunkte durch Zeilenumbruchmarken <br> konnte Tidy zwar XHTMLisieren, aber nicht in etwas Sinnvolles umwandeln. Hier muss man also von Hand eingreifen.

```
<br />
<img src="12x2_tp.gif" alt="" height="2" width="12"/>
<a href="angebote.html">Kursangebote</a>
<br />
```

Listing 8.8 Die Navigation enthält Platzhalter und Zeilenumbrüche.

Schritt 3 – Handarbeit

Das Logo wurde überarbeitet und verkleinert, um die obere Tabellenzeile niedriger gestalten zu können. Die Breiten- und Höhenangaben der Tabellenzellen und der Tabelle wurden entfernt, ebenso die Hintergrundgrafiken. Das `cellpadding`-Attribut wurde entfernt. Das `cellspacing`-Attribut der Tabelle wurde hingegen beibehalten, da es (derzeit) per CSS nicht befriedigend ersetzbar ist.

Die Tabellenzellen erhielten `id`-Attribute, um sie in den Rollen »Logo«, »Kopfbereich«, »Navigationsbereich« und »Inhalt« per CSS zu erfassen. Auf diesem Weg werden Abmessungen, Padding und gegebenenfalls Hintergründe wieder zugewiesen. Aus der Pseudoüberschrift im Kopfbereich wurde eine echte Überschrift `<h1>`, der zentrierende `<div>`-Container konnte als überflüssig entfallen:

```
<td id="kopfbereich"><h1>Kursangebote</h1></td>
```

Die Navigation wurde durch eine `<ul>`-Liste mit `id` ersetzt, womit gleich auch die Platzhaltergrafiken entfallen. Per CSS wird ein Hover-Effekt erzeugt.

```
<ul id="navigation">
...
  <li><a href="angebote.html">Kursangebote</a></li>
...
</ul>
```

Textformatierung, Farben, Hintergründe und Abmessungen werden durch ein verknüpftes Stylesheet festgelegt. Die Dateigröße der Seite schrumpfte trotz Beibehaltung der Layouttabelle von 8,4 KB auf 5,4 KB, was immer noch einer Reduzierung um rund 35 % entspricht.

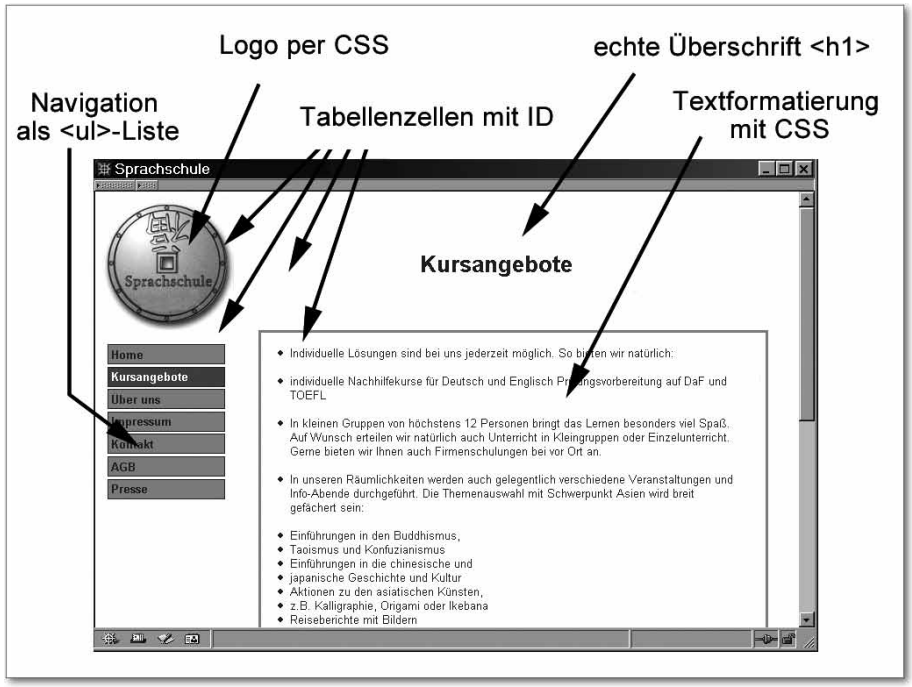

Abbildung 8.14 Tabellenlayout mit CSS verfeinert

Fazit

Die Möglichkeiten, eine Layouttabelle mit CSS zu formatieren, sind notwendigerweise eingeschränkt, da man an die Zeilen- und Spaltenstruktur gebunden bleibt. Als Notbehelf mögen solche Überarbeitungen vertretbar sein. Der Raum für Verbesserungen in Bezug auf Layout und Barrierefreiheit bleibt enorm. Ein `<div>`-Layout ist auf Grund seiner Flexibilität stets vorzuziehen.

8.4 Zusammenfassung

Aus diesem Kapitel sollten Sie folgende Kernaussagen mitnehmen:

- HTML Tidy ist ein Kommandozeilen-Tool, das für alle Betriebssysteme erhältlich ist.
- HTML Tidy ermöglicht die Überprüfung und Verbesserung von HTML-Quellcode.
- Per Stapelverarbeitung kann HTML Tidy alle HTML-Dokumente eines Verzeichnisses auf einmal verarbeiten.
- HTML Tidy kann Präsentationsmarkup entfernen oder durch CSS-Anweisungen ersetzen. Im Text erscheinende Sonderzeichen können automatisch durch Entities ersetzt werden. Durch Einrückungen wird gleichzeitig die Lesbarkeit des Quellcodes verbessert.
- HTML-Tidy ist in der Lage, auch fehlerhafte HTML-Dokumente in korrekte XHTML-Dokumente umzuwandeln und fügt Namensraum und Doctype-Deklaration hinzu.
- Verarbeitungsvorschriften für HTML Tidy lassen sich als Konfigurationsdateien ablegen und im Laufe einer Verarbeitung laden.
- HTML Tidy ist in zahlreichen Editoren bereits integriert. Darüber hinaus existieren auch Plug-in-Versionen oder autarke grafische Oberflächen.
- Während Codeoptimierung bei Überarbeitungen automatisiert werden kann, bedürfen Aspekte, die Layout, Barrierefreiheit und Semantik der Webseite betreffen, zu großen Teilen der Handarbeit.

Dem Design für mobile Clients sind allerhand technische Hürden entgegengesetzt. Dennoch ist der Entwurf von Webseiten für Minidisplays keine aussichtslose Angelegenheit, sondern durchaus reizvoll. Einige Randbedingungen wollen allerdings beachtet werden.

9 Webdesign für mobile Clients

Dem Webdesigner, der für Mobilgeräte entwerfen möchte, stellen sich vor allem die Limitierungen der Zielgeräte entgegen. Diese werden hauptsächlich durch die kleinen Abmessungen der Displays, die schwächeren Prozessoren, den kleineren Arbeitsspeicher sowie die reduzierten Möglichkeiten der Farb-, Grafik- und Schriftdarstellung verursacht. All diese Aspekte müssen folglich beim Design für entsprechende Clients berücksichtigt werden.

9.1 Welche Markup-Sprache für mobile Geräte?

Was das Grundgerüst von Webseiten für mobile Geräte angeht, herrscht inzwischen weitgehende Einigkeit, wenn auch nicht in allen Details. Die Zeiten der proprietären Sprachen wie HDL, cHTML und auch die der XML-Anwendung WML sind vorbei – gemeinsamer Standard ist inzwischen **XHTML**.

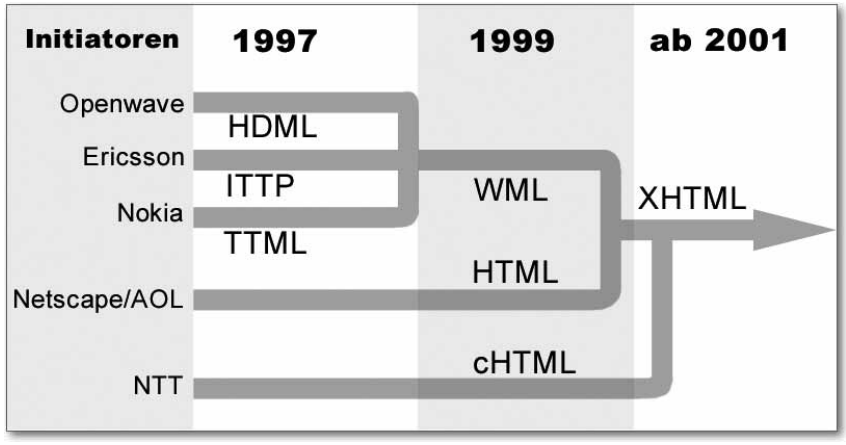

Abbildung 9.1 Die Entwicklung der Markup-Sprachen für Mobilgeräte

Zwar existiert hiervon auch eine von der **Open Mobile Alliance** propagierte XHTML-Variante **XHTML mobile profile**, jedoch können die meisten mobilen Clients auch normales XHTML konsumieren (im Gegenteil scheint XHTML MP auf manchen Clients eher problematisch).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
 "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
  <title>Rosen</title>
  <link href="mobil.css" rel="stylesheet" type="text/css"/>
</head>
<body>
  <h1>Rosen</h1>
  <p>Rosen (f., Singular Rose, lat./botanisch Rosa) sind
   Str&auml;ucher mit etwa 300 Arten aus der Familie der
   Rosengew&auml;chse.
   <img src="rose.jpg" alt="Bild einer Rose"/></p>
</body>
</html>
```

Listing 9.1 rosen_xhtml_mp.html

9.2 Technische Randbedingungen mobiler Geräte

Die **Prozessorleistung** mobiler Geräte ist aus Gründen der Stromersparnis gering und wird bei abnehmendem Batteriepegel eher noch weiter gedrosselt. Achten Sie daher darauf, dass Ihre Anwendung dem Mobilclient nicht zuviel Rechenleistung abverlangt. Dies betrifft auch die zu ladende **Datenmenge** einer Seite, welche die Speicherkapazität des Clients nicht übersteigen sollte.

Da, abgesehen von den genannten technischen Beschränkungen durch Arbeitsspeicher und Prozessor, auch (zumindest in den meisten Fällen) der Daten-Traffic teuer zu Buch schlägt, werden viele Nutzer den Download von Grafiken einschränken oder ganz unterbinden.

Dateneingabe

Neben dem kleinen Display besteht ein weiteres Manko mobiler Clients im Fehlen einer horizontalen Scrollfunktion. Im Unterschied zu Desktop- oder Laptop-Rechnern existiert für Mobilgeräte meist auch keine adäquate Eingabemöglichkeit per Tastatur (oder eine sehr rudimentäre). Stattdessen wird über eine **nume-**

rische Tastatur und anstelle einer Maus mit einem Griffel oder einem Eingabestift gearbeitet. Die Interaktionsmöglichkeiten unterscheiden sich daher notwendigerweise von denen bei Desktop-Rechnern. Beispielsweise ist die **Dateneingabe** mühsamer und somit langsamer. Vermeiden Sie es also, einem mobilen Nutzer überflüssige Schreibarbeit aufzubürden.

Erfolgt die Dateneingabe direkt per Stift oder Touchscreen, so handelt es sich bei diesem **Eingabegerät** dennoch nicht um etwas, einer Maus Vergleichbares. Die Stifteingabe kann noch als Klick-Event gewertet werden, Mouseover- und Mouseout-Ereignisse existieren jedoch nicht. Analog kann auch eine Tasteneingabe als Klick gewertet werden. Gehen Sie aber immer davon aus, dass Sie **keine vollständige Event-Unterstützung** zum Steuern von Scripten haben – keinesfalls sollten Sie sich auf Doppelklick-, Rechtsklick- oder Teil-Events wie Keydown und Keyup verlassen. Problemlos sind geräteunabhängige Events wie Blur und Focus.

Sonderfunktionen

Nicht dass Mobilgeräte ohne besondere Fähigkeiten wären, die bei der Erstellung von an sie gerichteten Webseiten Vorteile böten. Beispielsweise kann ein Mobilgerät über ein spezielles Linkprotokoll zum Absetzen einer **SMS** bewegt werden (auf einem Desktoprechner funktioniert dies selbstverständlich nicht):

```
<p><a href="sms:0049312345678">Schicken Sie eine SMS</a></p>
```

Hinweis: Speziell aus dem iMode-Bereich stammen einige interessante Erweiterungen für den Ankercontainer. Sie erfahren mehr über diese proprietären Attribute in der Referenz zum `<a>`-Tag in Abschnitt 14.1. **[«]**

9.3 Displays von Mobilgeräten

Die **Displaygrößen** mobiler Geräte variieren in einem Bereich von Briefmarkengröße bis zu halbwegs nennenswertem Screenestate. Die Auflösung in Pixeln, also Schärfe und Detailliertheit der Darstellung, ist dabei nicht vollständig an die physischen Abmessungen gebunden; manch scheinbar großzügig dimensioniertes Display zeichnet sich durch eine bemerkenswert kleine Pixelauflösung aus. Eine Obergrenze ist jedoch durch die Forderung der Mobilität gegeben: Ein Mobilgerät muss Taschenformat besitzen.

Als **Standardauflösungen** haben sich etabliert

- Hochformat 176 x 208 Pixel (kleinstes aktuelles Format)
- Hochformat 176 x 220 Pixel

- Querformat 240 x 160 Pixel
- Hochformat 240 x 260 Pixel
- Hochformat 240 x 320 Pixel (meistverbreitetes Format)
- Querformat 320 x 240 Pixel
- Hochformat 352 x 416 Pixel (Palmtop und vergleichbare Geräte)

Gelegentlich finden sich Geräte mit noch größeren Auflösungen bis zu 640 x 480 Pixel (z. B. T-Mobile Ameo), wobei es sich hier eher um abgespeckte Notebooks handelt. Ebenso können Sie auf ältere Geräte treffen, deren Display lediglich 160 x 100 Pixel auflöst (z. B. BlackBerry 6230). Rechnen Sie damit, dass für ständig eingeblendete Menüleisten weiterer Screenestate verlorengeht.

Die Unterscheidung zwischen Hoch- und Querformat ist nicht zwingend: Manche Geräte schalten das Display medienabhängig (beispielsweise zur Darstellung von Fotos) von Hoch- auf Querformat um.

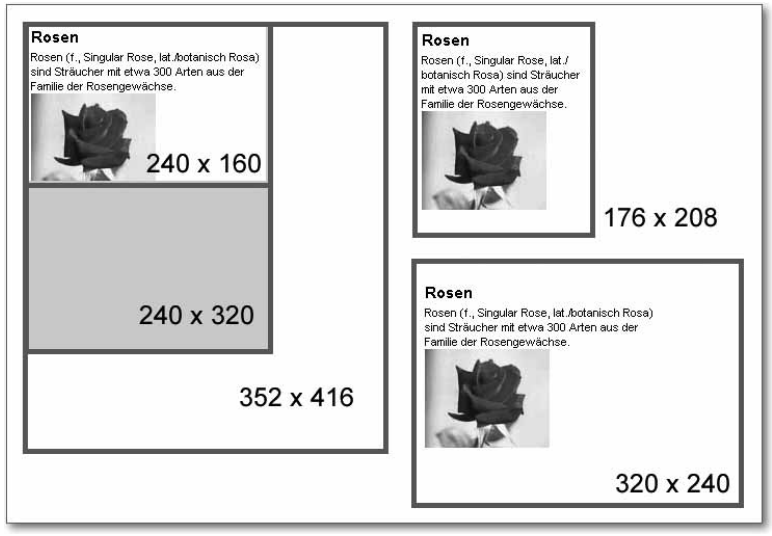

Abbildung 9.2 Displaygrößen für Mobilgeräte im Vergleich

9.4 Design für mobile Geräte

Die geringe Displaygröße bedeutet, dass der Designer platzsparend vorgehen muss. Generell wird dies auf ein einspaltiges, **linearisiertes Layout** hinauslaufen. Illustrationen, welche die zur Verfügung stehende Displaybreite überschreiten, sollten ausgeblendet werden (siehe Abbildung 9.3, **www.spiegel.de**).

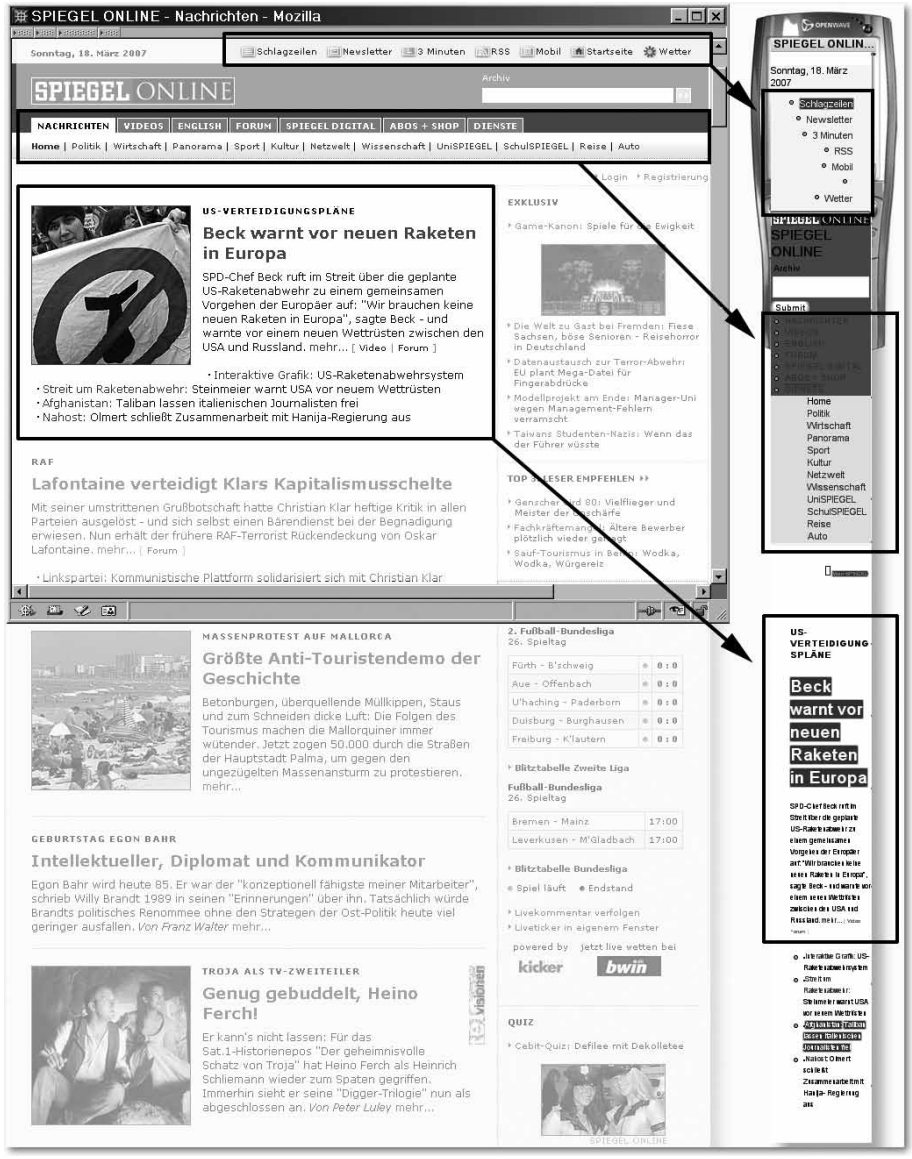

Abbildung 9.3 Durch Linearisierung werden Seiten eher länger

Als Faustregel gilt, dass Sie mit der Definition von Außen- und Innenabständen möglichst sparsam umgehen sollten. Zwischen Textabsätzen und Überschriften werden geringere Zwischenräume benötigt, Rahmen sollten auf die Breite von einem Pixel beschränkt werden oder ganz entfallen. Wenn es geht, sollten Sie auf **Darstellung in Quelltextreihenfolge** abzielen, also auf positionierte und gefloa-

tete Elemente verzichten. Wo Letztere noch geringere Schwierigkeiten machen (sie sind lediglich nicht besonders sinnvoll), können Positionierungen zu Überlagerungen, Verdeckungen oder zur Darstellung außerhalb des Displaybereichs führen. Bestenfalls muss dann gescrollt werden, schlimmstenfalls werden Inhalte unzugänglich.

Da horizontales Scrolling auf Mobilgeräten grundsätzlich problematisch oder nicht möglich ist, stellen auch Objekte mit festen Abmessungen ein ernsthaftes Problem dar. Hierzu gehören naheliegenderweise größere Grafiken, aber auch Formulareingabefelder. Setzen Sie per CSS ihre Maximalbreite (das `max-width`-Property) auf 100 % oder etwas weniger, um Probleme mit Overflow zu vermeiden.

Ebenfalls stellen sämtliche Elemente mit fixierter Position ein Hindernis dar (`position:fixed`). Sobald ein solches Element aus dem Darstellungsbereich hinausragt, ist es nicht mehr zugänglich.

9.4.1 Schriftgröße

Mobilgeräte verwenden kleinere Schriftgrößen und eine geringere Anzahl von Größenvarianten (Operas Small Screen Rendering kennt nur die drei Schriftgrößen »klein«, »mittel« und »groß«). Sie können daher nicht darauf bauen, dass sich Überschriften in ihrer Größe unterscheiden. Verzichten Sie bei der Größenangabe der Überschriften auf einen im Verhältnis zum normalen Fließtext stehenden Faktor, der größer als zwei ist.

Hinzu kommt die Tatsache, dass auf mobilen Geräten eine kleinere Zahl von Fonts, Schriftschnitten etc. zur Verfügung steht. Je weniger »exotisch« die Schrift Ihrer Wahl ist, desto näher wird die Darstellung auf einem Mobilgerät Ihrer ursprünglichen Intention kommen.

9.4.2 Quelltextaufbau

Die ohnehin erfolgende Linearisierung funktioniert dann am Besten, wenn die Struktur Ihres Quelltextes von vornherein darauf ausgelegt ist. Linearisierung bedeutet im Grunde einfach die Darstellung in Dokumentreihenfolge (was im Quelltext zuerst steht, erscheint auch zuerst auf dem Schirm). Sobald Tabellen im Spiel sind, wird die Lage schwieriger, da auch diese linearisiert werden.

An dieser Stelle muss an die »Above The Fold«-Regel erinnert werden: Displays von Mobilgeräten sind wesentlich kleiner als Desktop-Bildschirme und das Scrollen ist mühsamer. Außer einer Site-Überschrift (gegebenenfalls können Sie hier auch ein kleines Logo platzieren), der Sprungmarke zur Navigation und gegebe-

nenfalls einer Breadcrumb-Navigation, sollte nichts in Quelltextreihenfolge am Beginn der Seite vor den eigentlichen Inhalten stehen. Ausführliche Navigationsmenüs platzieren Sie nach dem Haupttext. Gehen Sie davon aus, dass kein Nutzer sich auf jeder Seite durch eine längere Navigation durcharbeiten möchte.

Textauszeichnung

Die Feinheiten der Textdarstellung gehen bei kleiner Schrift verloren. Verwenden Sie für Hervorhebungen daher besser Fettauszeichnung anstelle von kursiver Schrift (also `<b>` oder besser `<strong>` anstelle von `<i>` oder `<em>`). Die Openwave-Mobilbrowser haben aus diesem Grund von Haus aus die Eigenheit, die `<em>`-Marke nicht kursiv, sondern kursiv *und* fett umzusetzen. Dies gilt jedoch nicht für andere Hersteller.

Tabellen

Tabellen in HTML sind zeilenorientiert aufgebaut. Die Datenzellen werden entsprechend Zeile für Zeile von links nach rechts (was der Leserichtung von oben nach unten entspricht) linearisiert. Dies führt notwendigerweise dazu, dass der direkte Bezug zwischen vertikal übereinander stehenden Zellen verloren geht. Haben Sie beispielsweise Grafiken in einer Zeile platziert und die dazugehörigen Beschreibungstexte in der darunterliegenden Zeile, so wird dieser rein optische Bezug bei der Linearisierung auseinandergerissen.

Sehr unangenehm ist der Effekt, wenn Tabellen eingesetzt werden, um die Fragmente zerschnittener Grafiken zusammenzuhalten (beispielsweise für komplexe Seitenkopfgrafiken). In diesem Fall können Sie kaum damit rechnen, dass eine brauchbare Darstellung auf Mobilclients möglich ist. Da eine Dateigrößenoptimierung durch Zerschneiden von Grafiken bei heutigen Bandbreiten kaum noch eine Rolle spielt, weichen Sie besser auf (dateitechnisch meist nur unwesentlich größere) Grafiken in voller Größe aus. Diese können per CSS als Hintergrundgrafik eines Containers eingebunden werden, was es ermöglicht, die Grafik auf einem Mobilgerät erst gar nicht zu laden oder eine Alternativgrafik mit kleineren Abmessungen zu verwenden.

Bilder & Grafiken

Mobilgeräte können Grafiken zwar darstellen, sofern deren Abmessungen dies zulassen oder eine optische Skalierung möglich ist. Da auch im letzteren Fall die übertragene Datenmenge nicht geringer ist, tendieren viele Nutzer dazu, den Download von Grafiken von vornherein zu unterbinden.

Sie sollten deshalb in Erwägung ziehen, den Einsatz von Grafiken minimal zu halten und auf überflüssige Illustration zu verzichten. Nicht benötigte Grafiken kön-

nen Sie per `display:none` ausblenden. Mobilgeräte werden sie in diesem Fall nicht laden (die Bindung der Styleklasse an das Image-Objekt erfolgt, *bevor* die eigentliche Ressource beim Server angefragt wird).

Sofern Sie sich entscheiden, eine Grafik im Layout zu belassen, geben Sie mit den HTML-Attributen `height` und `width` deren Abmessung an. Dies erleichtert auch einem Mobilbrowser das Rendering. Sofern dies möglich ist, sollten Sie allerdings vorskalierte oder beschnittene Varianten Ihrer Grafiken anbieten, da das clientseitige Skalieren oft unbefriedigende Ergebnisse bringt (abgesehen vom vorhin angesprochenen Problem der Dateigröße). Sie sollten möglichst keine Grafiken verwenden, welche die Abmessungen überschreiten, die das Display des Zielgerätes unterstützt (versuchen Sie daher unter 240, besser unter 167 Pixel Breite zu bleiben).

Zumindest der Opera-Browser (auch in seinen Mobilversionen) kann dazu bewegt werden, anstelle eines Bildes dessen `alt`-Text darzustellen. Dergestalt auszublendende Bilder können Sie per Styleklasse bestimmen oder auch die Ausblendung für alle Grafiken pauschal vornehmen. Es genügt die Angabe:

```
img.alttext { content: attr(alt);}
```

Den dargestellten Text können Sie als Alternativtext kennzeichnen (geben Sie ihm beispielsweise eine Hintergrundfarbe oder einen Rahmen):

```
img.alttext { content: attr(alt); background:#ffff55;}
```

Stellen Sie also stets einen `alt`-Text zur Verfügung (dies ist zwar eigentlich obligatorisch, kann aber nicht oft genug wiederholt werden). Navigationen für Mobilgeräte sollten selbstverständlich nicht grafikbasiert sein, ebenso sollten Sie keine Grafiken anstelle von Überschriften oder Textblöcken einsetzen.

9.4.3 Farben

Die ersten mobilen Geräte besaßen monochrome Displays, waren aber auch nur in geringem Maße internetfähig. Inzwischen sind Farbdisplays Standard, wobei jedoch die Farbtiefe variiert – den Farbumfang eines Desktopgeräts erreichen Mobilgeräte nicht. Die Zeiten der 8-Bit-Farbgrafiken kommen dennoch nicht wieder: die **Farbtiefe** mobiler Geräte liegt in der Regel bei 262.144 Farben (es gibt derzeit allerdings nur wenige Geräte mit einer höheren Farbtiefe).

Aus der Farbtiefe darf nicht ohne Weiteres auf die Darstellungsqualität geschlossen werden. Der **Kontrast**, den ein Mobildisplay bietet, ist von Haus aus geringer als bei einem Desktopgerät und wird aus Gründen der Stromersparnis meist noch weiter zurückgeregelt. Zudem werden Mobilgeräte oft unter schwierigen

Lichtverhältnissen betrieben, was die Erkennbarkeit kontrastarmer Layouts zusätzlich erschwert. Die Faustregel der Barrierefreiheit, möglichst guten Kontrast zwischen Vorder- und Hintergrundfarbe zu bieten, gilt für das Mobildesign daher doppelt. Allzu subtil können Sie in Ihrer Farbwahl nicht sein.

Bestimmte Geräte setzen die **Textfarbe** ungeachtet Ihrer Angaben auf Schwarz (für Operas Small Screen Rendering trifft dies zu). Im Rückschluss ist es daher kontraproduktiv, Informationen an die Farbwiedergabe zu binden, also die Bedeutung eines Textes durch dessen Farbe zu unterstreichen. Dies ist allerdings ebenfalls eine allgemeine Forderung der Barrierefreiheit.

9.4.4 Navigation

Auch für Mobilgeräte sollten Navigationen auf Listen aufbauen. Obwohl nachfolgend geordnete Listen `<ol>` empfohlen werden, lässt sich das gleiche Ergebnis auch mit einer ungeordneten Liste `<ul>` erreichen. Sie müssten ihr in diesem Fall nur `list-style-type:numeric` zuweisen.

Es empfiehlt sich, beim Einsatz von **Accesskeys** in Menüs, die auf mobile Geräte ausgerichtet sind ausschließlich **Zahlenwerte** zu nutzen, da diese über die numerische Tastatur am einfachsten zu bedienen sind. Die Verwendung einer **geordneten Liste** zur Erstellung des Menüs bewirkt quasi »im Vorübergehen«, dass den Menüpunkten in Form der Listenbezifferung auch gleich der Accesskey vorangestellt wird (Sie dürfen dann natürlich nicht `list-style-type:none` angeben).

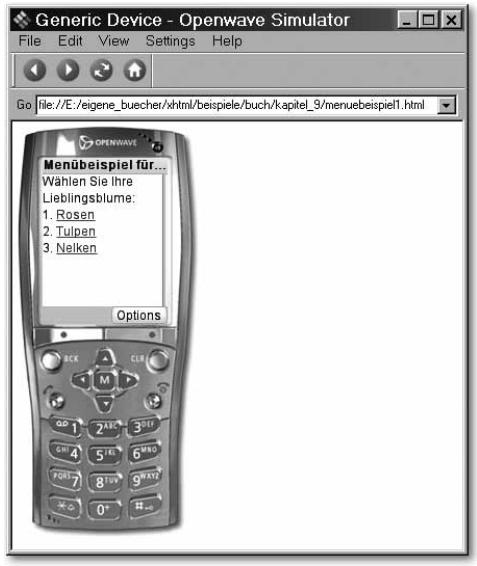

Abbildung 9.4 Das Menübeispiel im Openwave Phone Simulator

```
<p>W&auml;hlen Sie Ihre Lieblingsblume:</p>
<ol>
  <li><a href="rosen.html" accesskey="1">Rosen</a></li>
  <li><a href="tulpen.html" accesskey="2">Tulpen</a></li>
  <li><a href="nelken.html" accesskey="3">Nelken</a></li>
</ol>
```

Sie sehen diesen Quelltext in Abbildung 9.4 im Openwave Phone Simulator.

9.5 CSS für mobile Geräte

Das W3C empfiehlt für Mobilgeräte eine Untergruppe von CSS2, das **CSS Mobile Profile 1.0**. Die Unterschiede zu regulärem CSS sind verhältnismäßig gering. Berücksichtigt werden technische Unterschiede zu Desktop-Clients. CSS Mobile Profile 1.0 kennt daher beispielsweise keinen `:hover`-Selektor, da Mobilclients keine Maus-Ereignisse unterstützen. Auch Attributselektoren gehören nicht zum Repertoire der Mobilspezifikation, ebenso wenig die Pseudoelemente `:before` und `:after` oder Pseudoselektoren wie `:first-line` und `:first-letter`.

Tabellenbezogene Properties und einige zur Textgestaltung (wie `letter-spacing` oder `line-height`) fehlen, sowie weitere Eigenschaften, für Hintergrundgestaltung oder Positionierung. Verallgemeinert lässt sich sagen, dass die entfernten Properties für Mobilgeräte keine Rolle spielen. Die Spezifikation kann somit als Anhaltspunkt verwendet werden, welche CSS-Befehle anwendbar sind, auch wenn Sie nicht wirklich beabsichtigen, CSS gemäß des CSS Mobile Profiles zu schreiben (Sie können reguläres CSS 2.1 verwenden).

Der media-Attributwert »handheld«

Um Mobilgeräte zu veranlassen, nur die an sie gerichteten Styleanweisungen zu laden, können diese durch einen speziellen `media`-Wert gekennzeichnet werden: `media="handheld"`. Hierdurch werden die Anweisungen im Gegenzug von solchen Clients nicht berücksichtigt, für die sie nicht gedacht oder geeignet sind. Beachten Sie, dass dieses Stylesheet **zusätzlich zu einem allgemeinen Stylesheet** ohne `media`-Attribut oder mit `media="all"` wirksam wird, so dass Sie lediglich dessen Grundeinstellungen überschreiben bzw. ergänzen müssen.

Der mobile Client lädt das folgende Zusatz-Stylesheet:

```
<!-- Stylesheet für Mobilgeräte: -->
<link rel="stylesheet" href="mobil.css" media="handheld">
```

Wird ein ergänzendes `handheld`-Stylesheet geladen, so genügt es, Abmessungen und gegebenenfalls Farbkontraste zu kompensieren oder unerwünschte Inhalte

auszublenden (analog zu einem Druckstylesheet). Folgende Möglichkeiten haben Sie, um »handheld«-Styleanweisungen einzubinden:

- Externes Stylesheet mit `media`-Attribut:

```
<link rel="stylesheet" href="mobil.css" type="text/css"
      media="handheld">
```

- Dokumentstyle mit `media`-Attribut:

```
<style type="text/css" media="handheld">
  /* Anweisungen */
</style>
```

- Bedingter Block innerhalb eines Dokumentstyles, der nur von »handheld«-Geräten beachtet wird:

```
<style type="text/css">
  @media handheld { /* Anweisungen */ }
</style>
```

- Importanweisung mit `media`-Deklaration:

```
<style type="text/css">
  @import url('mobil.css') handheld;
</style>
```

9.5.1 Überlegungen zu einem Stylesheet für mobile Clients

Ergebnis der Anwendung eines »handheld«-Stylesheets soll eine linearisierte, einspaltige Darstellung der Webseite sein, die der Dokumentreihenfolge entspricht. Es ist im Prinzip gleichgültig, ob der Darstellung im Desktop-Browser ein Float- oder ein Positionslayout zugrunde liegt. Setzen Sie für das Mobil-Stylesheet Ihre **logischen Dokumentbereiche** auf `position:static` und überschreiben Sie alle Float-Anweisungen durch `float:none`.

Wenn Sie einen alle Inhalte umgebenden Wrapper-Container einsetzen, so setzen Sie dessen Breite auf 100 % und deaktivieren Sie eventuelle Margins, Paddings, Rahmen und Höhenangaben. Auch die logischen Bereiche benötigen keine Angaben für Höhe und Breite. Überschreiben Sie die Angaben aus dem Hauptstylesheet durch den Wert `"auto"`. Margins zum linken und rechten Displayrand können Sie im Wrapper-Container setzen (hier ist 1 % nach links und rechts angegeben):

```
/* Positionen und Breiten linearisieren: */
#container { width:100%;
             border:0;
             margin:0 1%;
             padding:0; }
```

```
/* exemplarische Bereichsbezeichner: */
#kopf, #navigation, #inhalt, #seitenfuss {
             position:static; float:none;
             width:auto; height:auto;
             border:0; margin:0; padding:0;}
```

Die Darstellung der **Menüs** soll ebenfalls linearisiert werden. Auf jeden Fall sollten horizontale, durch Floaten erzeugte Menüs entfloatet werden. Auch als Blockelemente definierte Ankercontainer sollten in ihren Normalzustand zurückversetzt werden, wobei auch der Unterstrich reaktiviert wird. (Sie können auch die Listenbullets wieder sichtbar machen, wenn dies zum Design passt.)

```
/* Gefloatete Menüs "entfloaten" (falls nötig): */
#navigation ul, #navigation li {
             float:none; }
/* Ankercontainer auf Inline setzen (falls nötig) */
#navigation a {
             display:inline;
             width:auto; height:auto; border:0;
             text-decoration:underline; }
```

Skiplinks am Dokumentbeginn sollten eingeblendet werden. Durch die Linearisierung wird die Seite sehr lang und Sie sollten dem User die Möglichkeit nicht vorenthalten, sofort zum Inhalt oder der Navigation weiterzugehen. Je nachdem, ob Sie Ihre Sprungmarken durch Positionierung versteckt haben (empfohlen) oder per `display:none` ausgeblendet haben (weniger günstig, da Sie sie auf diese Art auch vor Screenreadern verbergen), geben Sie etwa folgendes an:

```
#sprunglinks { position:static; display:block; }
```

Abbildungen sollten maximal die Displaybreite einnehmen. Dies kann pauschal über `max-width` bestimmt werden. Illustrationen, die rein dekorativen Zwecken dienen, können ganz entfallen (Sie könnten sie durch eine CSS-Klasse `"illu"` oder ähnlich kennzeichnen). Mobilgeräte werden sie dann nicht laden:

```
img      { max-width:100%; }
img.illu { display:none; }
```

Überschriften und Textabsätze erhalten einheitliche Zeilenhöhen und Margins. Eventuelle Paddingangaben werden genullt. Die Schrifthöhe kann individuell gesetzt werden. Sie können entweder einheitliche Größen für Überschriften angeben, oder variieren.

```
h1, h2, h3, h4, h5, h6, p {
          line-height: 1em; margin: 1.3em 0; padding:0}
h1, h2, h3 { font-size: 1.23em; }
h4, h5, h6, p { font-size:0.9em; }
```

Bestimmen Sie die **Schriftfamilie** mit einer allgemeinen Angabe wie »serif« oder »sans-serif«. Spezifischer zu werden lohnt kaum, da Mobilgeräte von sich aus den lesbarsten Font wählen. Dies kann global im `<body>` geschehen:

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

Tabellen können dadurch linearisiert werden, dass alle Tabellenelemente auf Blockverhalten umdefiniert werden (das Ergebnis ist Geschmackssache, siehe Abbildung 9.5). Gleichzeitig kann eine allgemeine Schriftgröße gesetzt werden:

```
table, thead, tbody, tfoot, tr, td, th {
            display: block; font-size:1em; }
```

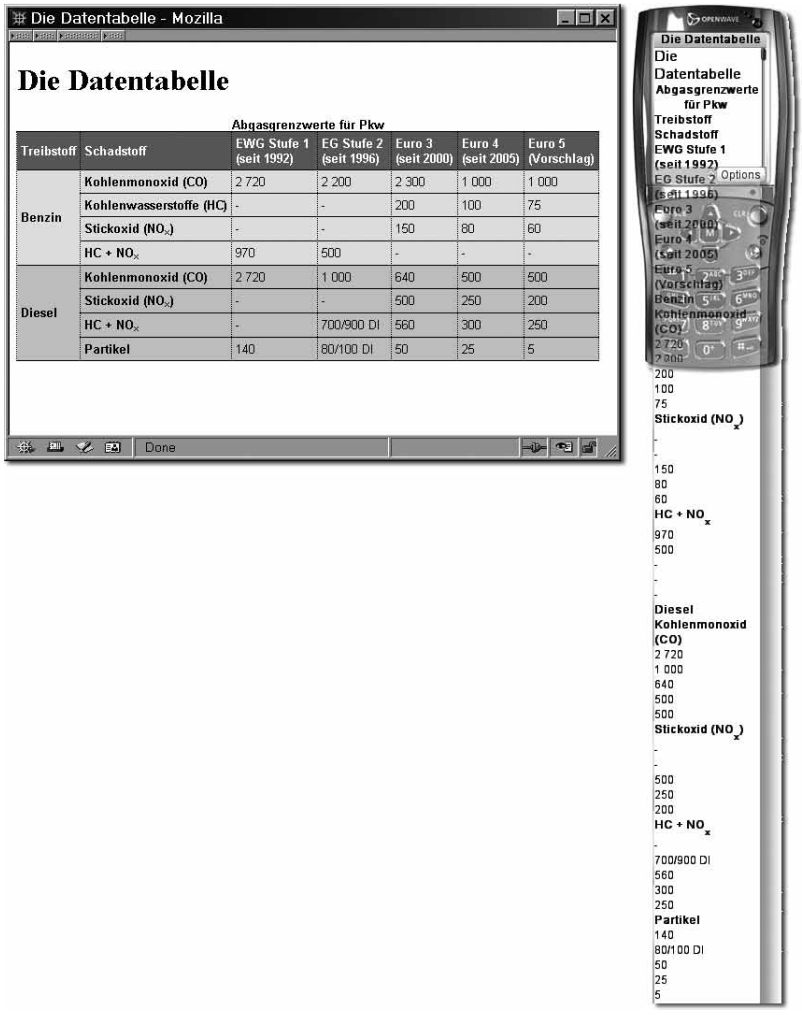

Abbildung 9.5 Die linearisierte Tabelle wird unhandlich lang.

Formulare müssen ebenfalls linearisiert werden. Günstiger ist es, wenn in diesem Zusammenhang keine Layouttabellen verwendet wurden. Setzen Sie auf jeden Fall alle Margins, Paddings und Rahmen der Formularelemente auf Null.

```
form, fieldset {margin:0; border:0; padding:0};
```

Die Eingabefelder müssen, um ins Display zu passen, über `max-width` eine Maximalbreite erhalten (siehe Abbildung 9.6):

```
input, textarea, select {max-width: 100%; }
```

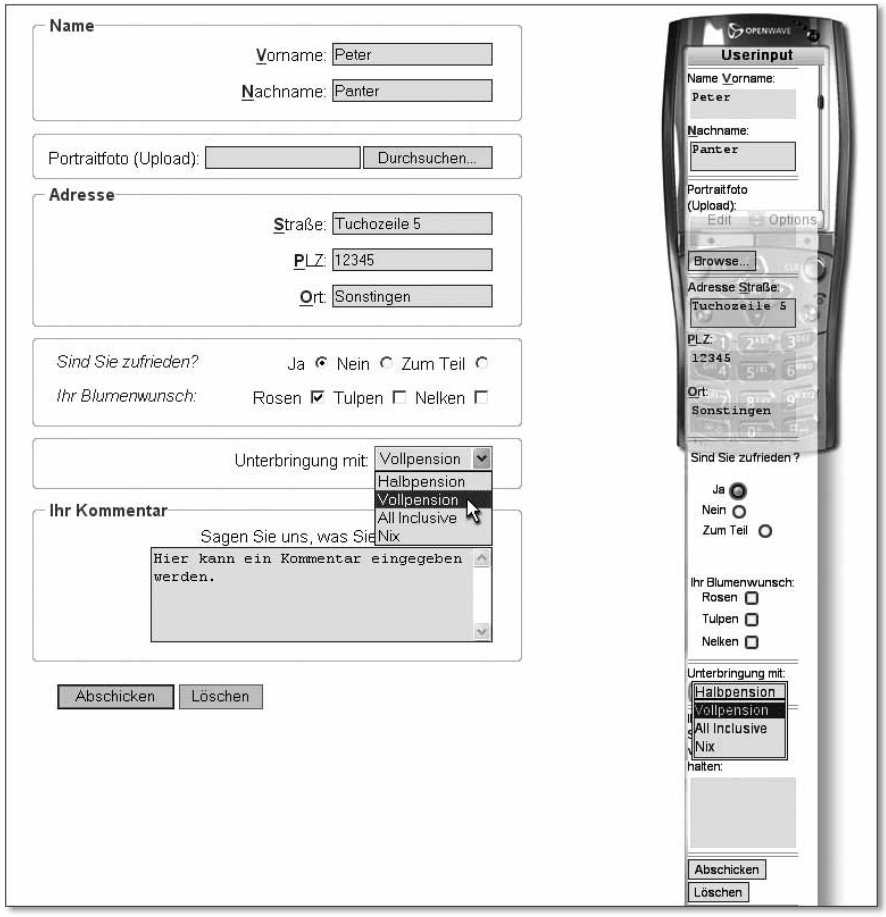

Abbildung 9.6 Das linearisierte Formular erfordert viel Scrolling

Für **Listen** können Regeln analog zu Textabsätzen definiert werden. Dies betrifft die Schriftgröße (setzen Sie Margins und Padding nach Geschmack):

```
li, dt, dd { font-size: 1em;}
```

Für Mobil-Layouts gilt, wie bei jedem Layout für Desktop-Browser: Testen, testen, testen. Günstigenfalls haben Sie ein Gerät in der Form Ihrer Zielplattform zur Verfügung. Ansonsten können Sie sich mit Simulationen behelfen.

9.6 Test von Mobil-Designs mit Opera

Mit gewissen Abstrichen (es handelt sich schließlich nicht um ein echtes Mobilgerät) lässt sich auch ein Desktop-Opera zum Test der Tauglichkeit eines Layouts für Mobilgeräte verwenden. Sie können dafür entweder den **Normalmodus** mit entsprechend kleinem Viewport einsetzen oder den **Small Screen Rendering Modus**. (Für Mozilla/Firefox ist ein vergleichbarer Darstellungsmodus als Option der Webdeveloper Toolbar verfügbar).

Abbildung 9.7 Aktivieren von Operas Small Screen Rendering

9.6.1 Test in Normalansicht

Der Normalmodus ist stets dann zu wählen, wenn Sie ein CSS-Stylesheet für »handheld devices« eingebunden haben. Opera verwendet in diesem Fall das Small Screen Rendering nicht, hält sich also an die Anweisungen des »handheld«-Stylesheets.

Bringen Sie zunächst das Browserfenster, also Ihren Viewport, auf die Größe des Zielgeräts. Sie können in Opera unter **Extras > Einstellungen > Erweitert** festlegen, dass die aktuelle Fenstergröße in der Titelleiste des Fensters angezeigt wird (Abbildung 9.8).

Abbildung 9.8 Operas Einstellungsdialog

Anschließend steht Ihnen die Information über die Displaygröße in der Titelleiste zur Verfügung (Abbildung 9.9). Arbeiten Sie mit einer Fenstergröße von 240 Pixel Breite und 320 Pixel Höhe, um die meist verbreitete Displaygröße zu simulieren.

Abbildung 9.9 Opera mit 240x320 Viewport, Scrollbars deaktiviert

Um dichter an die Darstellung eines Mobilgeräts heranzukommen, können Sie typische Desktop-Browser-Funktionalitäten deaktivieren. Opera bietet Tastenkürzel, um **Adressfeld** oder **Scrollbalken** ein- und auszuschalten. Probieren Sie aus, ob die Seite ohne Verwendung einer Maus zugänglich bleibt. Darüber hinaus sollten Sie Ihr Mobil-Layout auch mit abgeschaltetem JavaScript oder ohne Grafiken ausprobieren. Sieht in diesem Fall immer noch alles brauchbar aus, so gilt dies auch für Mobilgeräte.

Funktion (Opera)	Tastenkürzel
Small Screen Modus	⇧ + F11
Adressfeld ausblenden	Strg + F8
Scrollbalken deaktivieren	Strg + F7
JavaScript deaktivieren	F12 und J

Tabelle 9.1 Operas Tastenkürzel für Testoptionen

9.6.2 Test in Small-Screen Rendering-Ansicht

Da die meisten aktuellen Webseiten nicht für Mobilgeräte optimiert sind, bietet der im Mobilsektor verbreitete Opera-Browser ein als **Small Screen Rendering** bezeichnetes Feature an, das anzuzeigende Webseiten umarrangiert, sodass sie auf kleinen Displays darstellbar sind. Das Small Screen Rendering wird nur aktiv, wenn kein Style mit `media="handheld"` vorhanden ist.

Opera stellt dieses Feature auch innerhalb seiner Desktopversion zur Verfügung. Der Browser eignet sich daher in gewissem Rahmen zum Testen von Mobilanwendungen.

Folgende **Umformungen** nimmt das Small Screen Rendering vor:

- **Linearisierung der Seite**
  Die Seite wird als **einspaltiges Layout** dargestellt. Beruht das Original auf Tabellenlayout oder einem mehrspaltigen `<div>`-Layout, so werden Zellen- oder Spalteninhalte in Quelltextreihenfolge angeordnet.
- **Skalierung der Inhalte**
  Bilder und andere Objekte werden skaliert, wenn sie die **Breite der Darstellungsfläche** überschreiten. Auf diesem Weg wird die Notwendigkeit des horizontalen Scrollens vermieden. Bilder, die nicht erfolgreich skaliert werden können, werden aus der Darstellung entfernt. Kompositbilder, die in Desktop-Browsern durch Übereinanderlegung von Teilgrafiken dargestellt werden, versagen in diesem Modus. Dasselbe gilt für zerschnittene Grafiken (»sliced images«).
- **Skalierung der Schriftgrößen**
  Da in SSR (wie bereits erwähnt) **nur drei Schriftgrößen** zur Verfügung stehen, erfolgt eine Nivellierung der Schriftgrößenvarianten der Seite. Wenn Ihre Website in dieser Ansicht funktioniert, können Sie in Erwägung ziehen, auf weitere Maßnahmen wie spezielle »handheld«-Stylesheets zu verzichten.

Abbildung 9.10 Small Screen Rendering (Ansicht »Kleinbildschirm«) in Opera

9.7 Test von Mobil-Designs mit Openwave Phone Simulator

Für Windows 2000 und XP existiert eine Simulations- und Testumgebung von Openwave (**www.openwave.com**), der **Openwave Phone Simulator 7.0**. Die Umgebung emuliert den Openwave Mobile Browser 7.0. Sie kann mit WML ebenso umgehen, wie mit XHTML, CSS (allerdings nicht `media="handheld"`) und dem von Openwave propagierten XHTML MP.

Die Software stellt eine Protokoll- und Debugging-Konsole und die Emulation der Oberfläche eines Mobiltelefons zur Verfügung, in der Webseiten betrachtet werden können. Sie können sowohl über **File > Open File** lokale Dateien betrachten, als auch unmittelbar Webadressen in das Adressfeld eingeben.

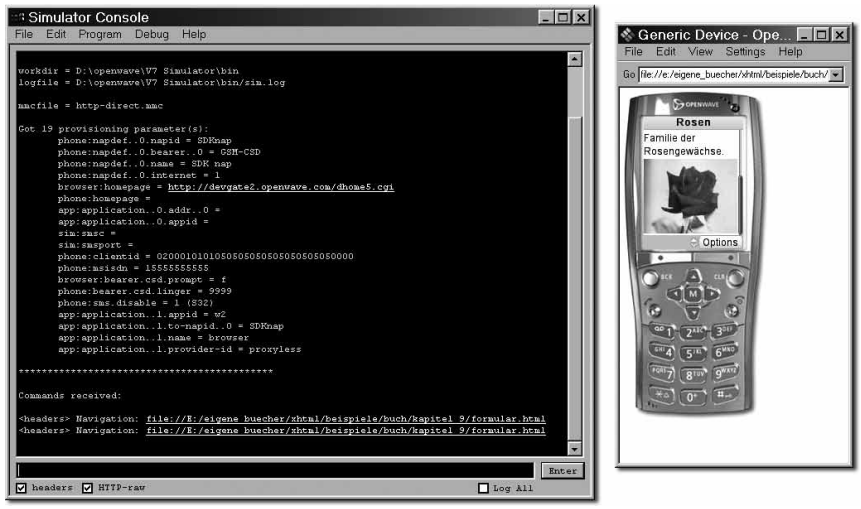

Abbildung 9.11 Openwave Phone Simulator Konsole (links) und Device (rechts)

Sie erhalten den Phone Simulator bei Openwave unter folgender Adresse:

- **http://developer.openwave.com/dvl/tools_and_sdk/phone_simulator/**

9.8 Zusammenfassung

Was Sie aus diesem Kapitel mitnehmen sollten:

- Heutige Mobilgeräte können mit XHTML und CSS arbeiten, unterliegen jedoch prinzipbedingten technischen Limitationen, zu denen Displaygröße, Arbeitsspeicher und Rechenleistung gehören.
- Beim Design für Mobilgeräte muss darauf geachtet werden, dass die Quelltextstruktur eine Linearisierung der Darstellung erlaubt. Verwenden Sie daher keine Layouttabellen.
- Berücksichtigen Sie, dass durch eine Linearisierung die Seiten länger werden. Blenden Sie Sprungnavigationen für Mobilgeräte stets ein.
- Berücksichtigen Sie beim Formularentwurf, dass die Dateneingabe auf einem Mobilgerät mühsamer ist.
- Formulare und Tabellen müssen linearisierbar sein. Inhalte wie Formulareingabefelder, Bilder oder Objekte dürfen die Displaybreite nicht überschreiten.
- Mobilgeräten kann im Prinzip über `media="handheld"` ein eigenes Stylesheet zur Verfügung gestellt werden. Prüfen Sie jedoch, ob Ihr Zielgerät dies auch unterstützt.
- Eliminieren Sie für ein Mobil-Stylesheet alle Positionierungen und Floats und setzen Sie Margins und Paddings auf Minimalwerte.
- Testen Sie auf echten Geräten oder verwenden Sie Simulationen wie Operas Small Screen Rendering oder den Openwave Phone Simulator.

Wie geht es weiter?

Das folgende Kapitel 10, *Barrierefreiheit und Accessibility*, fasst die in den vorangegangenen Kapitel eingestreuten Bemerkungen zur Barrierefreiheit zusammen und ergänzt sie um allgemeine Hinweise und Beispiele.

Dieses Kapitel führt in die Grundlagen des Entwurfs barrierefreier Websites ein und bietet Ihnen neben Quellcodebeispielen auch erläuterte Checklisten zum Prüfen und Einstufen der Barrierefreiheit.

10 Barrierefreiheit und Accessibility

Unter **Accessibility** versteht man die Zugänglichkeit eines Webangebots für seine Nutzer. Tauchen hierbei in keinem Fall Hindernisse für die vollständige Nutzung auf, so spricht man von **Barrierefreiheit**.

10.1 Barrierefreiheit – für wen?

Gewöhnlich werden die beiden Begriffe Barrierefreiheit und Accessibilty synonym verwendet, wobei »Accessibility« das Gewicht gefühlsmäßig stärker auf technische und »Barrierefreiheit« eher auf menschliche Aspekte legen mag. Das Internet wird als Informationsquelle immer wichtiger und sollte keine Nutzer ausschließen – hierfür muss auch der Webdesigner Sorge tragen.

Abbildung 10.1 Standardansicht einer Website (www.jazzdimensions.de)

10.1.1 Warum barrierefrei programmieren?

Eine barrierefreie Programmierung, welche die breitestmögliche Nutzung einer Website erlaubt, ist keine Generosität einer Randgruppe[1] gegenüber, sondern sollte eine Selbstverständlichkeit sein. Jeder Betreiber einer Website muss es als ureigenes Interesse ansehen, dass seine Inhalte jedermann zugänglich sind, ob nun aus rein kommerziellen oder anderen Gründen.

Abbildung 10.2 Ansicht mit vergrößerter Schrift weiß auf schwarz (Prototyp)

Vorteile barrierefreier Programmierung:

- Zugang für visuell, motorisch oder kognitiv gehandicapte Benutzer
- Zugang für verschiedene (auch mobile) Endgeräte
- Zugang für alte Browserversionen
- Kleinere Datenmenge, daher Unterstützung langsamer Internetanbindungen

1 In den USA liegt der Anteil an Schwerbehinderung bei der Bevölkerung bei 18 %. Von den jungen Menschen unter 18 Jahren sind ca. 7 %, von den Senioren ab 65 Jahren über 38 % schwerbehindert. Quelle: www.cdc.gov/nchs/. In Deutschland gelten ca. 8 % der Gesamtbevölkerung als schwerbehindert. Quelle: www.destatis.de.

- Leichtere Sitewartung durch Trennung von Layout und Inhalt
- Erleichterung der Katalogisierung für Suchmaschinen

Der Gesetzgeber sagt …

Geht es speziell um die Nutzung von Webangeboten durch Nutzer mit Handicaps, so greifen auch verschiedene Gesetzgebungen und Vorschriften. Dies gilt primär für Websites, die von der öffentlichen Hand (Bund, Lander, Kommunen) betrieben werden – hier besteht seit dem Jahr 2002 die Verpflichtung, entsprechende Angebote barrierefrei zu gestalten. Grundlage ist das *Behindertengleichstellungsgesetz (BGG)* und vergleichbare Gesetze auf Landesebene.

Details hierzu sind in der *Barrierefreie Informationstechnik-Verordnung (BITV)*[2] geregelt, die in Anlehnung an die US-amerikanische *Web Accessibility Initiative (WAI)*[3] erstellt wurde. Das W3C definiert in seinen *Web Content Accessibility Guidelines (WCAG)*[4] drei Unterstützungsgrade A, AA und AAA mit jeweils wachsenden Ansprüchen an die Zugänglichkeit.

Nicht verpflichtend – aber sinnvoll

Eine allgemeine gesetzliche Verpflichtung zur Erfüllung der Forderungen laut BITV oder WAI existiert in Deutschland (noch) nicht. Dennoch ist eine überprüfende Orientierung zumindest anhand der ersten beiden der drei in der BITV unterschiedenen Prioritätsstufen für jeden Websitebetreiber sinnvoll.

Keine Barrierefreiheit, sondern »nur« Barrierearmut

Eine vollständige Barrierefreiheit im Sinne des Wortes kann es im Internet nicht geben – stets wird die eine oder die andere Nutzergruppe bestimmte Webangebote nur teilweise, unzureichend oder gar nicht nutzen können, stets werden sich in der einen oder anderen Form technische (Soft- und Hardware) oder persönliche (mangelnde Kenntnisse oder Handicaps der Nutzer) Hindernisse in den Weg stellen. Allenfalls ist eine *Barrierearmut* zu erreichen, die aber auch angestrebt werden sollte. Gleichzeitig wäre es natürlich unbefriedigend, wenn diese Barrierearmut durch Reduktion auf den kleinsten gemeinsamen Nenner aller potentiellen Nutzergruppen erreicht würde.

Welchen Nutzergruppen stellen sich Barrieren entgegen?

Das Standardbild als Argument für barrierefreie Programmierung, der »blinde oder sehbehinderte Nutzer« mit Screenreader oder Braille-Display mag plakativ

2 Wortlaut der BITV: http://bundesrecht.juris.de/bitv/index.html.

3 Die WAI im Internet: http://www.w3.org/WAI/.

4 WCAG 2.0 beim W3C: http://www.w3.org/TR/WCAG20/.

sein. Das Auftreten von Barrieren bei der Nutzung von Inhalten stets ausschließlich mit Behinderungen der Nutzer zu verbinden, ist jedoch falsch.

Barriere-Situationen

- Der Nutzer kann bestimmte Inhalte (Text, Bilder, Videos) nicht sehen.
- Der Nutzer sieht die Inhalte, aber nicht in der gewünschten Reihenfolge (Linearisierungsproblem).
- Der Nutzer kann eingebettete Audiodaten nicht anhören.
- Der Nutzer kann bestimmte Funktionen (z. B. Formulare) nicht nutzen.
- Der Nutzer kann im Webangebot nicht navigieren.

Nicht in jedem Fall ist ein barrierebildender Umstand auf ein Handicap des Nutzers zurückzuführen – auch das Ausgabegerät und seine Peripherie müssen in Betracht gezogen werden.

Das Ausgabegerät als Barriere

- Nicht jedes Ausgabegerät verfügt über einen Bildschirm oder über einen Bildschirm mit ausreichender Größe, Farbdarstellung etc.
- Nicht jedes Ausgabegerät kann mit Frames umgehen.
- Nicht jedes Ausgabegerät kann Audio- oder Videodaten wiedergeben.
- Nicht jedes Ausgabegerät verfügt über eine Maus oder eine Tastatur.
- Nicht jedes Ausgabegerät verfügt über einen Scriptinterpreter, der z. B. JavaScript ausführen kann.

Ein Internetnutzer ist heutzutage nicht auf einen Bildschirm und einen Webbrowser als Ausgabemedium beschränkt – auch ein Autoradio, das einem notwendigerweise beschäftigten Fahrer Webseiten vorliest, kommt einem Screenreader gleich. Ein Nutzer, der Webseiten über sein Mobiltelefon abruft, hat weder einen normalgroßen Bildschirm noch Tastatur und Maus zur Verfügung, ähnliches gilt für Nutzer von PDAs. Zu berücksichtigen sind ferner die nichtmenschlichen Nutzer des Internets, die Suchmaschinen, die auf die Inhalte einer Website ebenfalls textorientiert zugreifen.

Eine barrierearm konzipierte Website berücksichtigt all diese Aspekte quasi automatisch, indem sie allen potentiellen User Agents und damit ihren Nutzern die Informationen in einer für sie zugänglichen Form bereitstellt. Dies erfordert gegebenenfalls auch die Bereitstellung alternativer Inhalte, beispielsweise Erläuterungen in Textform anstelle von Bildern oder Videos.

10.1.2 Technologien zur Überwindung von Barrieren

Beinahe jede Barriere kann mit einer geeigneten Technologie bzw. einem entsprechend programmierten User Agent abgemildert, umgangen oder überwunden werden. Das Internet ist stark visuell ausgerichtet, daher tritt die häufigste Barriere dadurch auf, dass zu übermittelnde Information an Bilder geknüpft ist oder nur durch Bilder geschehen soll. In dieser Richtung existieren auch die meisten technologischen Lösungen.

Für die Überwindung der visuellen Barriere dienen *Screenreader*, die die Inhalte in Textform auswerten und akustisch wiedergeben – eine alternative Ausgabemethode findet man in den *Braille-Displays*, die den Text in taktil aufnehmbare Form umwandeln. Für die Wiedergabe der Inhalte von Grafiken oder Videos sind solche User Agents auf alternative Textbeschreibungen angewiesen. *Bildschirmlupen* mildern die Barriere lediglich, indem sie die die Darstellung der Inhalte vergrößern.

Abbildung 10.3 Ausschnittsvergrößerung mit Bildschirmlupe

Für Audiodateien muss nicht notwendigerweise nach einer technologischen Lösung gesucht werden – hier kann ebenfalls eine beschreibende oder den Inhalt wiedergebende Textalternative vorgesehen werden. Speziell an gehörlose Nutzer

würde sich eine Umsetzung gesprochener Inhalte in Gebärdensprache richten, was allerdings aufwendig zu realisieren ist.

Bereits an dieser Stelle wird deutlich, dass eine Überwindung von Barrieren oft nur mit Hilfe des Webdesigners möglich ist. Eine Nutzung eines Webangebots scheitert dann, wenn nicht die Grundvoraussetzungen für Barrierearmut berücksichtigt wurden.

Hilfsmittel für barrierefreies Webdesign

Als Webdesigner können Sie sich die Arbeit erleichtern, indem Sie auf einige kostenlos zur Verfügung gestellte Hilfsmittel zurückgreifen. An erster Stelle zu nennen ist hier die **Web Accessibility Toolbar** für den Internet Explorer zu nennen, die im (englischsprachigen) Original von der australischen Organisation »Vision Australia« angeboten wird (**www.visionaustralia.org.au**). An gleicher Stelle steht auch der **Color Contrast Analyzer** zur Verfügung, der Farbkontraste einer Webseite beurteilen und bewerten hilft.

Die Web Accessibility Toolbar auf Deutsch

Die Web Accessibility Toolbar ist ebenfalls in eingedeutschter Version »DE 1.2« über die Website von »Web for all« erhältlich (**www.webforall.info**). Das Gleiche gilt für den Color Contrast Analyzer.

Als kommerzielles Produkt bietet **UsableNet** (**www.usablenet.com**) das Dreamweaver-Plug-in **Lift** an (aktuell Version 2.2), das Projekte in Dreamweaver gemäß den Vorschriften von W3C und Section 508 barrierefrei umsetzen hilft. Der gleiche Hersteller stellt kostenfrei (mit geringerem Funktionsumfang) auch die Dreamweaver-Erweiterungen »Accessibility Suite for W3C/WCAG« und »508 Accessibility Suite« zur Verfügung (Registrierung erforderlich).

10.2 Das Einmaleins der Barrierearmut

Es gibt einige Faustregeln, die einen guten Schritt auf dem Weg zu einer barrierearmen Website darstellen, wenn sie beherzigt werden. Ein Teil von ihnen betrifft Grafiken und Alternativtexte zu diesen Grafiken sowie Erleichterungen der Navigation – dies lässt sich auch nachträglich in herkömmlich gestaltete Websites einarbeiten und verbessert deren Zugänglichkeit.

Andere Regeln jedoch betreffen bereits den Grundaufbau der Seite auf HTML-Ebene. Letztere können für ältere Sites einen erheblichen Umbau bedeuten; für neu erstellte Sites ist ihre Beachtung jedoch (fast) problemlos: Eine korrekt

geschriebene, auf XHTML und CSS basierende Webseite ist von vornherein nahezu barrierefrei.

Frühzeitig an Barrierefreiheit denken

Stets sollte die Berücksichtigung von Aspekten der Barrierefreiheit bereits in der Konzeptionsphase der Websiteerstellung geschehen – dies nachträglich einzuarbeiten, ist aufwendig und teuer.

Um eine Seite auf ihre Barrierearmut zu untersuchen, können Sie online eine BITV-Selbstbewertung durchführen, und die Einhaltung der BITV-Regeln überprüfen (kostenfreie Registrierung erforderlich). Geprüft werden die in 14 Gruppen geordneten 66 Anforderungen, die laut BITV durch eine vollständig barrierefreie Website erfüllt sein müssen.

- **BITV-Selbstbewertung: www.bitvtest.de**

Abbildung 10.4 Prüfschritte zum Test der Barrierefreiheit (www.bitvtest.de)

10.2.1 Seitenaufbau und Layout

Eine wirklich barrierefreie Webseite funktioniert auch ohne CSS. Die in diesem Fall sichtbare lineare Struktur entspricht der Reihenfolge, in der textbasierte Browser und Suchmaschinen die Seite einlesen. Für solche User Agents ist die optische Präsentation unerheblich.

Seitenaufbau und inhaltliche Struktur

Der Aufbau von Seiten ist so zu gestalten, dass sie auch ohne Stylesheets oder mit User Agents ohne CSS-Unterstützung genutzt werden können. Folgt der inhaltliche Aufbau einer grundsätzlichen Logik, so wird er auch in diesem Fall in einer sinnvollen Reihenfolge dargestellt (WAI 6.1, BITV 6.1).

Praxistipp

Betrachten Sie Ihre Seite mit einem Textbrowser (Lynx oder Firefox mit Lynxviewer-Plug-in[5], um dies zu kontrollieren. Achten Sie speziell auf positionierte Inhalte: Bleiben diese ohne CSS in einer sinnvollen Reihenfolge?

Kurzcheck zu Seitenaufbau und Layout

- Vergeben Sie über `<title>` aussagekräftige Seitentitel.
- Achten Sie auf ein konsistentes Layout für die gesamte Internetpräsenz.
- Gestalten Sie Navigationselemente und -mechanismen einheitlich, mit konstanten Symbolen, gleichbleibender Nomenklatur und Position.
- Folgen Sie, sofern möglich, etablierten Konventionen bei der Benennung und Positionierung funktionaler Elemente (Homelink, Suchfeld, globale Navigation).
- Setzen Sie für spaltenorientiertes Layout CSS-Formatierungen ein und keine Frames oder Layouttabellen.

Keine Layouttabellen einsetzen

Das Layout der Webseite wird über CSS vorgenommen. Auf keinen Fall sollten Sie Layouttabellen einsetzen – dies kann erstens Unzugänglichkeit für bestimmte User Agents zur Folge haben, zweitens Probleme bei der Linearisierung für eine Ausgabe im Screenreader. Auch die BITV lehnt Layouttabellen aus diesen Gründen ab (WAI 5.3, BITV 5.3).

Praxistipp

Ersetzen Sie Layouttabellen durch CSS-Layout. Tabellen sollen lediglich eingesetzt werden, wo es um die tabellarische Darstellung von Daten geht. Wenn Sie erleben wollen, wie ein Screenreader ein Tabellenlayout interpretiert (bzw. dies versucht), so besuchen Sie eine entsprechende Seite mit dem Firefox-Plug-in »Fangs« (**http://sourceforge.net/projects/fangs**).

5 Firefox-Lynxviewer von Yellowpipe: www.yellowpipe.com.

Kurzcheck zu Layouttabellen

- Verwenden Sie keine Layouttabellen für Ihr Seitenlayout, sondern stets CSS-Anweisungen.
- Sollten Sie dennoch Layouttabellen einsetzen wollen oder müssen, achten Sie auf eine korrekte Linearisierung der Inhalte (inhaltlich logische Abfolge in Quelltextreihenfolge).
- Verzichten Sie bei Layouttabellen auf Tabellenstrukturelemente wie Tabellenheader `<th>`, Tabellenüberschriften `<caption>` sowie Zeilen- und Spaltengruppierungscontainer.
- Achten Sie auf korrektes Markup von Textinhalten innerhalb der Layoutzellen, verwenden Sie Überschriften, Textabsätze, Listen usw. (d.h. belassen Sie nie Text ohne Struktur innerhalb von Layoutzellen).

Kurzcheck zu Frames

Frames sind auf allen User Agents mit kleinem Viewport (Darstellungsfeld) problematisch, namentlich also auf Mobilgeräten wie PDAs und Mobiltelefonen. Zwar können auch solche Geräte (zumindest teilweise) mit Frames umgehen, dennoch leidet die Navigierbarkeit der Site, da meist nur eine der Inhaltsseiten des Framesets auf einmal dargestellt wird. Andere User Agents wie Suchmaschinen haben mit Frames notorische Schwierigkeiten, da die Verknüpfungen zwischen den Dokumenten nicht über eine `href`-, sondern eine `src`-Relation erfolgt, Verbindungen zwischen Seiten also nicht erkannt werden.

- Verwenden Sie, wenn dies möglich ist, keine Frames.
- Sofern Sie Frames einsetzen müssen, stellen Sie zum Frameset einen alternativen Inhalt mit `<noframes>` zur Verfügung, den beispielsweise Mobilgeräte und Suchmaschinen nutzen können.
- Bennenen Sie die Frames mit dem `name`-Attribut auf eine Weise, die ihre Position und Bedeutung eindeutig erkennbar macht. Auch dies ist für Mobilgeräte notwendig, die Frames als Liste zugänglich machen.
- Stellen Sie Informationen über den Inhalt der Frames zur Verfügung (hierfür kann das `longdesc`-Attribut eingesetzt werden).

10.2.2 Verwendung von Schriften und Farben

Wenn Sie die zu vermittelnde Information über das Design stellen (was bei einer barrierefreien Website vorrangig sein sollte), so ist Lesbarkeit oberstes Gebot. Dies betrifft sowohl die Wahl der Schrift als auch die Schriftgröße, -farbe und den Kontrast zum Texthintergrund. Generell sollten Sie all dies über CSS festlegen.

Kurzcheck zu Schriften

- Verzichten Sie auf HTML-Präsentationselemente wie `<font>` oder `<basefont>` zur Steuerung der Schriftdarstellung.
- Verwenden Sie CSS-Angaben für Schriftart, -größe und -farbe.
- Verwenden Sie möglichst wenige unterschiedliche Fonts auf einer Seite.
- Geben Sie Schriftgrößen in relativen Einheiten wie Prozent oder em an und setzen Sie die Mindestgröße entsprechend 11 pt.
- Verändern Sie nicht den Standardabstand zwischen den Buchstaben.
- Verwenden Sie zur Bildschirmausgabe serifenlose Schriften und Serifenschriften zur Druckausgabe. (Diese Forderung ist strittig und zu gewissem Teil Geschmackssache – verbreitet ist die Ansicht, Serifenschrift sei auf Papier lesbarer. Ob man für den Bildschirm serifenlose Schrift vorziehen sollte, hängt jedoch auch von der Sprache des Textes ab.)

Sinnvoller Einsatz von Farben und Kontrasten

Text sollte leserlich sein, sich also ausreichend vom Seitenhintergrund abheben. Vermeiden Sie zu schwache Kontraste. Bedenken Sie besonders beim Einsatz von Grafiken oder der Gestaltung Ihrer Navigation, dass manche Nutzer bestimmte Farbtöne nicht unterscheiden können: Der Informationsinhalt darf sich nicht auf Farben stützen (WAI 2.1, BITV 2.1).

Praxistipp

Vermeiden Sie beispielsweise die Verwendung von Rot und Grün als Kontrastfarben, da dies für Farbfehlsichtige zum Problem wird. Prüfen Sie ihr Layout mit einem Tool wie dem **Color Contrast Analyzer** auf ausreichenden Kontrast zwischen Vorder- und Hintergrundfarbe. Die **Web Accessibility Toolbar** bietet zudem die Möglichkeit, eine Webseite in Schwarz-Weiß zu betrachten. Schneidet Ihre Seite hierbei gut ab, so werden Sie auch bei der Druckausgabe durch eine bessere Optik belohnt.

Kurzcheck zu Farben

Farben und Farbkontraste sind kein reines Designthema, sondern spielen auch für die Barrierefreiheit eine Rolle, sobald ein potenzieller Nutzer ein Sichthandicap hat, also beispielsweise stark kurzsichtig oder farbenblind ist. Sie können diesen Nutzern entgegenkommen, indem Sie die Möglichkeit zur Wahl der Schriftgröße oder des Kontrasts bieten (siehe Abbildung 10.5). Stellen Sie hierfür alternative Stylesheets zur Verfügung.

- Achten Sie auf genügend Kontrast zwischen Text- und Hintergrundfarbe.
- Verzichten Sie auf Rot-Grün-Kontraste (betrifft farbenblinde Nutzer).

- Machen Sie die Erkennbarkeit von Information nicht von Farben abhängig.
- Verwenden Sie keine HTML-Attribute zur Farbgebung.
- Verwenden Sie für die CSS-Farbdefinition RGB- oder Hexadezimalwerte anstelle von Farbnamen.

Abbildung 10.5 Schriftgröße und Kontrastwahl bei www.einfach-otto.de

10.2.3 Quelltext und Formatierung

Grundlage einer barrierefreien Website ist stets korrekt geschriebenes HTML bzw. XHTML. Sofern Sie alle Elemente in ihrer semantischen Bedeutung und in vorgesehener Hierarchie einsetzen, haben Sie bereits den Großteil der in Folge vorgestellten Forderungen an den Quelltext erfüllt.

Korrekt und zweckdienlich eingesetztes (X)HTML

Der HTML-Code sollte den Spezifikationen entsprechen und validierbar sein. Gleichzeitig sollte Markup zweckdienlich eingesetzt werden, um Überschriften, Abkürzungen und vergleichbare Inhalte kenntlich zu machen (BITV 3.1 – 3.7).

Praxistipp

Sofern Sie korrektes (X)HTML einsetzen, ist dieser Punkt zu großen Teilen automatisch erfüllt. Machen Sie darüber hinaus Gebrauch von Elementen wie `<abbr>` und `<acronym>` und verwenden Sie Markup semantisch richtig (Überschriften, Listen, Tabellen). Meiden Sie die Zweckentfremdung von Elementen aus Präsentationsgründen (nehmen Sie `<blockquote>` zur Kennzeichnung von Zitaten und nicht zur Einrückung von Text).

Kurzcheck zum Markup von Inhalten

- Verwenden Sie Überschriften in korrekter hierarchischer Folge, ohne Hierarchiestufen zu überspringen (erst `<h1>`, dann `<h2>` usw.).

- Verwenden Sie HTML-Elemente und -Attribute korrekt, also beispielsweise den `<p>`-Container zur Kennzeichnung von Textabsätzen.
- Verzichten Sie auf präsentative Elemente oder Attribute (Textausrichtung) und steuern Sie dies über CSS.
- Verwenden Sie logische Elemente wie `<em>` oder `<strong>` zur Kennzeichnung von Hervorhebungen im Sinne von Betonungen.
- Verzichten Sie auf rein präsentative Hervorhebungen mit `<i>` oder `<b>`.
- Kennzeichnen Sie Zitate mit `<blockquote>` oder `<q>` und zweckentfremden Sie diese Elemente nicht (beispielsweise für Texteinzug).
- Kennzeichnen Sie Akronyme und Abkürzungen mit `<acronym>` und `<abbr>` in Zusammenhang mit deren `title`-Attribut.
- Verwenden Sie logische Markupelemente, wo sie angebracht sind (`<cite>`, `<code>`, `<del>`, `<ins>`, `<samp>`, `<var>` etc.).
- Setzen Sie keine blinkenden Texte ein (keinesfalls `<blink>`-Tag verwenden; wenn Sie, warum auch immer, dennoch blinkenden Text benötigen, verwenden Sie das entsprechende CSS-Property, damit das Blinken notfalls deaktiviert werden kann).

Kurzcheck zu Sprache

Eine Sprachangabe erleichtert einem Screenreader die Wahl eines Aussprachemodus für die Wiedergabe. Zudem wird Suchmaschinen die sprachbezogene Katalogisierung ermöglicht.

- Kennzeichnen Sie die Sprache des Dokuments (beispielsweise mit dem `lang`- oder `xml:lang`-Attribut im `<html>`-Starttag).
- Kennzeichnen Sie eventuelle Sprachwechsel innerhalb des Dokuments mit dem `lang`- oder `xml:lang`-Attribut.

Kurzcheck zu Listen

- Setzen Sie HTML-Markup für Listen korrekt ein und verwenden Sie es nicht für rein präsentative Zwecke (z. B. Texteinzug).
- Verschachteln Sie hierarchische Listen mit korrekter HTML-Syntax.
- Geben Sie grafischen Bullets ein `alt`-Attribut oder (besser) verwenden Sie an deren Stelle ein CSS-`list-style-image`.
- Legen Sie keine Zusatzinformation über Listenelemente in die Bulletgrafiken (z. B. über ihre Farbe oder Form).

10.2.4 Barrierefreie Tabellen

Tabellen stellen grundsätzliche Hürden dar, da sie nur schwer linearisierbar sind, ohne dass dabei Bezüge zwischen Zellinhalten verloren gehen. Um diese Bezüge zu verdeutlichen und zu erhalten, und auch zur generellen Unterstützung der Barrierefreiheit haben Sie spezielle HTML-Attribute zur Verfügung.

Attribut	Element	Aufgabe
`summary`	`<table>`	Enthält eine inhaltliche Beschreibung der Tabelle
`abbr`	`<td>`, `<th>`	Abkürzung des Zellinhalts
`axis`	`<td>`, `<th>`	Inhaltliche Kategorien, denen die Zellen zugeordnet werden können (Bedeutungsachsen)
`headers`	`<td>`	Verweis auf IDs einer oder mehrerer Headerzellen
`id`	`<th>`	Kennzeichnung einer Headerzelle als Ziel des `headers`-Attributs einer Datenzelle
`scope`	`<th>`	Zeilen-, spalten- oder gruppenorientierte Zuordnung von Datenzellen zur betreffenden Headerzelle

Tabelle 10.1 Barrierefreiheits-Attribute von Tabellenelementen

Eine Tabelle sollte eine Überschrift mit `<caption>` erhalten und in Tabellenkopf `<thead>` und -körper `<tbody>` unterteilt sein. Gegebenenfalls können auch mehrere `<tbody>`-Container vorhanden sein. Wird ein Tabellenfuß `<tfoot>` benötigt, so muss er nach dem Tabellenkopf und vor dem ersten `<tbody>`-Container eingefügt werden. Die Grundstruktur einer korrekt aufgebauten Datentabelle sieht daher folgendermaßen aus:

```
<table summary="Beschreibung des Inhalts der Tabelle">
<caption>&Uuml;berschrift f&uuml;r die Tabelle</caption>
<thead>
   <!-- hier die Kopfzeile und deren Headerzellen -->
</thead>
<tfoot>
   <!-- hier der Tabellenfu&szlig; und sein Inhalt -->
</tfoot>
<tbody>
   <!-- hier der Tabellenkörper mit den Inhalten -->
</tbody>
</table>
```

Im Tabellenkopf befinden sich die Headerzellen `<th>`. Diese erhalten eine beschreibende Abkürzung mit dem `abbr`-Attribut, einen Bezug zu einem Datenzellenbereich mit dem `scope`-Attribut und eine eindeutige Kennzeichnung mit dem `id`-Attribut. Hier ein Beispiel:

```
<thead>
   <tr>
   <th scope="col" id="stadt" abbr="Stadt">
    Name der Stadt</th>
   <th scope="col" id="kfz" abbr="Kfz-Kennzeichen">
    Zugeh&ouml;riges KFZ-Kennzeichen</th>
   <th scope="col" id="pkw" abbr="Zahl PKWs">
    Zahl der zugelassenen PKW</th>
   </tr>
</thead>
```

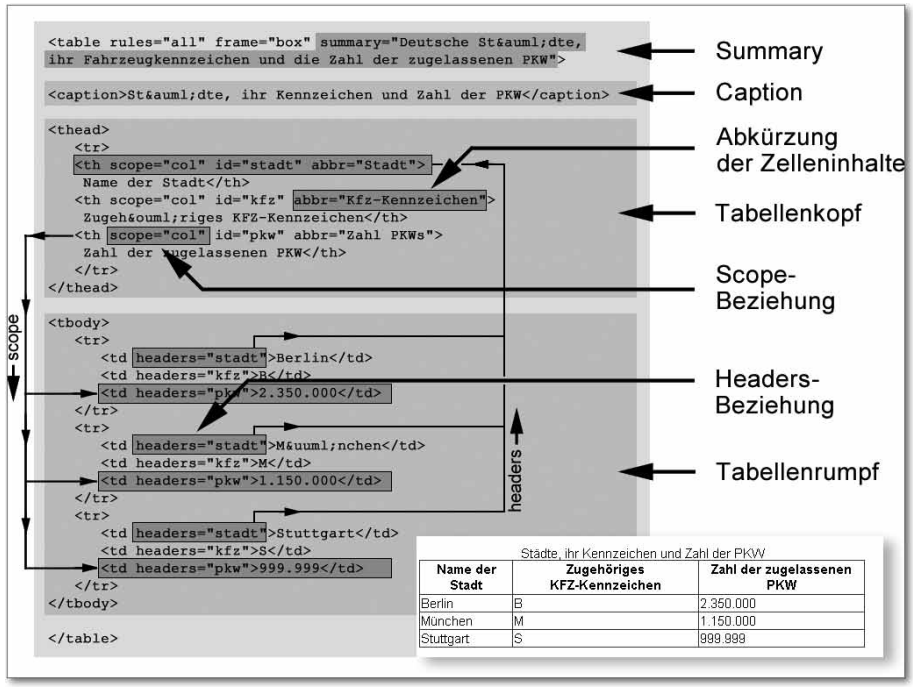

Abbildung 10.6 Schema einer barrierefreien Tabelle

Auf die IDs der Headerzellen wird aus dem Datenbereich durch ein gleichnamiges `headers`-Attribut Bezug genommen (die Definition der Beziehung in der anderen Richtung durch `scope` könnte im Prinzip jedoch bereits ausreichen):

```
<tbody>
   <tr>
      <td headers="stadt">Berlin</td>
      <td headers="kfz">B</td>
      <td headers="pkw">2.350.000</td>
   </tr>
```

```
    <tr>
       <td headers="stadt">M&uuml;nchen</td>
       <td headers="kfz">M</td>
       <td headers="pkw">1.150.000</td>
    </tr>
    <tr>
       <td headers="stadt">Stuttgart</td>
       <td headers="kfz">S</td>
       <td headers="pkw">999.999</td>
    </tr>
</tbody>
```

Kurzcheck zu Datentabellen

- Stellen Sie mit dem `summary`-Attribut eine Inhaltsbeschreibung der Tabelle zur Verfügung.
- Fügen Sie einer Datentabelle eine Tabellenüberschrift mit `<caption>` hinzu.
- Verwenden Sie Zeilen- und Spaltenüberschriften, denen Sie gegebenenfalls mit dem `abbr`-Attribut kurze Inhaltsangeben hinzufügen.
- Setzen Sie die Attribute `scope` oder `headers` und `axis` ein, um die Zeilen- und Spaltenüberschriften den Datenzellen zuzuordnen.
- Verwenden Sie Zeilengruppen wie `<thead>` und `<tfoot>`, wenn Sie umfangreiche Tabellen gestalten.
- Tabellieren Sie nicht mit dem `<pre>`-Element, da dies eine rein visuelle Präsentation darstellt, die sich verschiedenen User Agents nicht erschließt.

10.2.5 Links und Navigation

Mit dem Navigieren innerhalb des Informationsangebots einer Website steht und fällt deren Nutzwert. Barrierefrei sollte daher nicht nur die Hauptnavigation, sondern auch die »Ad hoc«-Navigation innerhalb der Seiteninhalte sein. Bewährt hat sich der Aufbau von Navigationsmenüs auf Basis von HTML-Listen vom Typ `<ul>`, deren optische Darstellung von CSS gesteuert wird und entsprechend Freiheiten lässt. Andere förderliche Maßnahmen wie das Bereitstellen von Sprunglinks und Ankerpunkten als Linkzielen setzen ebenfalls bereits in der Quelltextstruktur an.

Verständliche und barrierefreie Navigation

Die Navigation einer Website sollte konsistent und verständlich sein. Möglichst sollten Mechanismen zur Verfügung gestellt werden, die ihre Bedienung unabhängig von bestimmten Eingabegeräten macht (BITV 9.2). Linktexte sollten aus-

sagekräftig sein. Den Nutzern soll eine Orientierung über den aktuellen Standort und Hilfe in Form einer Sitemap geboten werden (BITV 13.1 – 13.4).

Praxistipp

Machen Sie Ihre Navigation deutlich als solche kenntlich und benennen Sie die Navigationspunkte eindeutig. Sofern Konventionen existieren, folgen Sie diesen ruhig (ein Warenkorb sollte »Warenkorb« heißen) – zu große Originalität an dieser Stelle kann kontraproduktiv sein. Verwenden Sie das `title`-Attribut für Links. Hyperlinks im Fließtext müssen als solche kenntlich sein – wenn Sie den Unterstrich deaktivieren, heben Sie einen Hyperlink auf andere Weise hervor.

Abbildung 10.7 Listenbasierte Navigation bei www.einfach-otto.de

Eine potentiell barrierefreie Navigation setzt nicht allein auf Maus-Interaktion, sondern stellt für eine **Tastaturbedienung** auch Tabindex und Accesskeys zur Verfügung. Im folgenden Beispiel werden die Buchstaben, die als Keyboardshortcut eingesetzt werden, im Linktext zusätzlich in `<span>`-Container eingeschlossen. Dies erlaubt es, sie bei Bedarf zusätzlich hervorzuheben. Dies kann im Rahmen eines alternativen Stylesheets geschehen oder grundsätzlich bereits in der Normalansicht. Folgendes CSS zeigt den Accesskey mit Unterstreichung. Für den restlichen Linktext ist der Unterstrich entfernt:

```
div#navigation a      { text-decoration:none; }
div#navigation a span { /* Hervorhebung:     */
                        textdecoration:underline; }
```

Hier der Quellcode einer listenbasierten, barrierefreien Navigation:

```
<div id="navigation">
   <ul>
      <li class="current">
      <a href="index.html" accesskey="h" tabindex="1">
      <span>H</span>ome</a></li>
      <li><a href="wir.html" accesskey="u" tabindex="1">
      &Uuml;ber <span>u</span>ns</a></li>
      <li><a href="katalog.html" accesskey="k" tabindex="1">
      <span>K</span>atalog</a></li>
      <li><a href="ref.html" accesskey="r" tabindex="1">
      <span>R</span>eferenzen</a></li>
      <li><a href="fbck.html" accesskey="f" tabindex="1">
      <span>F</span>eedback</a></li>
      <li><a href="imp.html" accesskey="i" tabindex="1">
      <span>I</span>mpressum</a></li>
   </ul>
</div>
```

Kurzcheck zur Navigation

Zum Erstellen einer barrierefreien Navigation und generell zur Erhöhung der Nutzbarkeit Ihrer Site sollten Sie die folgenden Regeln in Betracht ziehen. Sofern Sie keine Suchfunktion anbieten, ist eine Sitemap bei umfangreicheren Websites unabdingbar. Machen Sie diese möglichst leicht auffindbar (über die globale Navigation). Ebenso wichtig sind die Sprunglinks am Seitenanfang zu Navigation und Inhalt, die für normale Browser per CSS ausgeblendet werden.

- Bauen Sie Navigationsmenüs auf HTML-Listen auf.
- Platzieren Sie die Navigation konstant und im Sichtbereich und machen Sie die klickbare Fläche ausreichend groß.
- Benennen Sie Ihre Navigationspunkte eindeutig und selbsterklärend.
- Beschreiben Sie Linkziele mittels des `title`-Attributs des `<a>`-Tags.
- Gliedern Sie die Menüpunkte per `tabindex`-Attribut in die Tab-Reihenfolge ein.
- Geben Sie alternativ oder zusätzlich zum `tabindex`-Attribut die Möglichkeit der Tastatursteuerung über das `accesskey`-Attribut.

- Setzen Sie für ein horizontal angeordnetes Menü nie Ankerelemente unmittelbar nebeneinander, sondern trennen Sie sie stets durch Leerzeichen und weitere eingeschobene Zeichen (beispielsweise das |-Zeichen). Verlassen Sie sich nicht auf die optische Separierung der Links durch CSS-Anweisungen wie `margin` – Screenreader werten diese nicht aus.
- Bieten Sie eine Sitemap oder ein Inhaltsverzeichnis an.
- Bieten Sie am Seitenbeginn Sprunglinks zu Navigation, Seiteninhalt und weiteren wichtigen Punkten an
- Geben Sie dem Nutzer Orientierung über seine aktuelle Position in der Website (»Sie befinden sich hier«-Indikator).

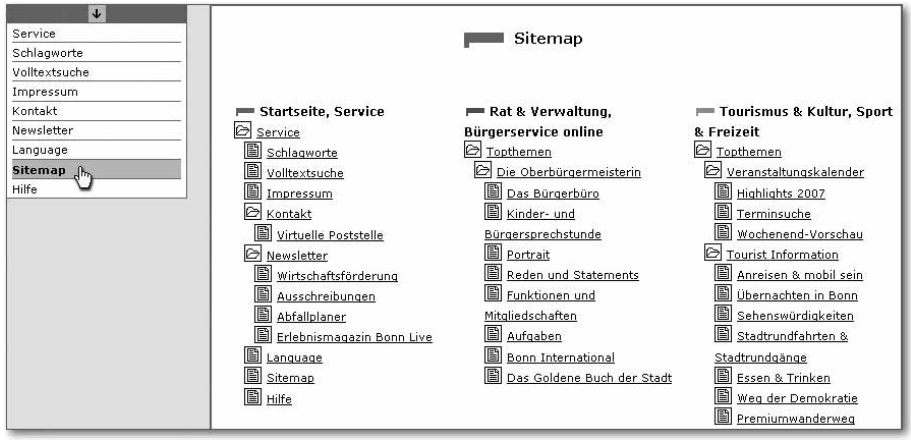

Abbildung 10.8 Vorbildliche Sitemap bei www.bonn.de (Auszug)

Kurzcheck zu Links im Textinhalt

Im Gegensatz zu Links in Navigationsmenüs, deren Aufgabe bereits durch den Kontext deutlich ist, müssen Links in Fließtexten hervorstechen, um als solche erkannt zu werden. Die einfachste Methode, dies zu erreichen besteht darin, die Defaultpräsentation mit Unterstrich in Texten beizubehalten. Icons (per CSS eingeblendet) können Zusatzinformationen über Art und Ziel des Links geben.

- Machen Sie Hyperlinks im Fließtext erkennbar. Sofern Sie den Linkunterstrich deaktiviert haben, heben Sie Links auf andere Art sichtbar hervor.
- Machen Sie deutlich, wenn Links auf externe Quellen zeigen oder in einem neuen Fenster geöffnet werden.
- Machen Sie deutlich, wenn Links auf eine Nicht-HTML-Ressource zeigen.

Abbildung 10.9 Externer Link durch Icon gekennzeichnet, oben bei Mouseover

An dieser Stelle ist der Link im Ausgangszustand nur durch Icon und Farbe gekennzeichnet. Dies ist zu vertreten, da es sich um eine erweiterte Linkliste handelt. Im Fließtext sollte dagegen der Unterstrich nicht entfernt werden. Bei Mausover gibt eine Hintergrundfarbe und eine gepunktete untere und obere Rahmenlinie optische Rückmeldung:

```
a.extern {
      background:#fff url('extern.gif') 0% 50% no-repeat;
      padding:1px 0 3px 15px;
      color:#D13500;
      text-decoration:none; }

a.extern:hover {
      background-color:#ffccbb;
      border-top:1px dotted #E13500;
      border-bottom:1px dotted #E13500; }
```

10.2.6 Barrierefreie Formulare

Formulare sind relativ leicht barrierefrei zu gestalten – Barrierefreiheit tritt praktisch automatisch ein, sobald man bestimmte Grundregeln der Gestaltung beachtet. Zu den wichtigsten gehört, dass Eingabefelder und deren Beschriftung einander problemlos zugeordnet werden können. Beide sollten darüber hinaus durch `<label>`-Elemente miteinander verknüpft werden. Die Formularfelder können zusätzlich einen Tabindex oder Accesskey erhalten – keine schlechte Idee ist es, ihnen per `title`-Attribut einen Erklärungstext hinzuzufügen, der als Tooltip bei Mouseover erscheint (siehe Abbildung 10.10 beim Feld »Lieferadresse«).

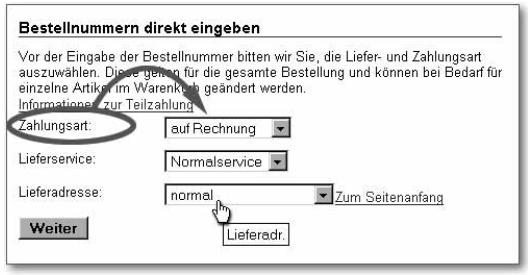

Abbildung 10.10 Barrierefreies Formular bei www.einfach-otto.de

Hier sehen Sie einen Quelltextausschnitt des in Abbildung 10.10 gezeigten Formulars. Achten Sie auf die Beziehung zwischen Label und Selectbox sowie auf das `title`-Attribut des Eingabeelements. Die erste Option ist vorselektiert.

```
<label for="zahlung">Zahlungsart:</label>
<select name="zahlung" title="Zahlungsart" id="zahlung">
   <option value="1" selected="selected">
    auf Rechnung
   </option>
   ...
</select>
```

Beispiel – Grundstruktur eines barrierefreien Formulars

Ein potentiell barrierefreies Formular sollte strukturiert sein. Überlegen Sie sich, ob Sie die einzugebenden Daten in logische Gruppen einordnen können und fassen Sie die Felder in Feldgruppen zusammen. Vergeben Sie einen Feldgruppenbezeichner mit `<legend>`, wo es sinnvoll erscheint.

Bauen Sie zunächst mit `<fieldset>`-Elementen eine **Struktur** auf und kennzeichnen Sie die Elemente mit IDs oder weisen Sie ihnen eine Stylesheetklasse zu. Dies erleichtert es nachfolgend auf ihre Inhalte gezielt mit CSS zuzugreifen. In diesem Beispiel verwende ich eine Klasse `"radiobox"` für Feldgruppen, die aus Checkboxen oder Radiobuttons bestehen und eine weitere Klasse `"submitgroup"` für die Feldgruppe, die Submit- und Resetbutton enthalten soll. Feldgruppen, die ausschließlich normale Eingabefelder, Selectboxen oder Textareas enthalten, bekommen keine Klasse.

Die Grundstruktur eines komplexeren Formulars könnte somit etwa folgendermaßen aufgebaut werden:

```
<form ...>
   <fieldset><legend>Name</legend>
      <!-- Eingabe des Namens -->
```

```
   </fieldset>
   <fieldset>
      <!-- Dateiupload für Foto o.ä. -->
   </fieldset>
   <fieldset><legend>Adresse</legend>
      <!-- Eingabe der Adresse -->
   </fieldset>
   <fieldset class="radiobox">
      <!-- Diverse Checkboxen und Radiobuttons -->
   </fieldset>
   <fieldset>
      <!-- Selectbox -->
   </fieldset>
   <fieldset>
      <!-- Textareafeld -->
   </fieldset>
   <fieldset class="submitgroup">
      <!-- Submit- und Resetbuttons -->
   </fieldset>
</form>
```

Beispiel – Eingabeelemente und Label

Grundsätzlich bekommt jedes Eingabeelement (mit Ausnahme von Submit- und Resetbutton) ein `<label>`-Element. Die `<input>`-Elemente benötigen daher stets ein `id`-Attribut und bekommen für den Datenversand ein zu diesem gleichnamiges `name`-Attribut.

Den `<label>`-Container um sein Eingabefeld herumzulegen, erleichtert das anschließende Stylen mit CSS. Eine weitere Aufgabe des `<label>` ist das Tragen der `tabindex`- und `accesskey`-Attribute. Das Zeichen für den Keyboardshortcut wird in der Beschreibung durch ein `<span>`-Element markiert, um es gegebenenfalls hervorheben zu können. Hier ein Beispiel:

```
<label for="Vorname" tabindex="1" accesskey="v">
   <span>V</span>orname:
   <input type="text" name="Vorname" id="Vorname" />
</label>
```

Funktionale Gruppen (so genannte »Arrays«) von Radiobuttons oder Checkboxen benötigen ein gemeinsames `name`-Attribut, aber individuelle `id`-Attribute, um ihnen Einzellabels zuweisen zu können. Das Buttonarray wird hier zusätzlich noch in einen `<div>`-Container eingeschlossen (dies kann unterbleiben, wenn Sie nur eine solche Gruppe pro Feldgruppenelement haben).

```
<div>
   <label for="rg1_r1">Ja
      <input type="radio" name="radgruppe1"
       id="rg1_r1" value="r1" checked="checked"/>
   </label>
   <label for="rg1_r2">Nein
      <input type="radio" name="radgruppe1"
       id="rg1_r2" value="r2" />
   </label>
   ...
</div>
```

Das Verfahren für Dateiuplodfelder, Textareas und Selectboxen entspricht dem für normale Eingabefelder. Das im vorangegangenen Quelltextbeispiel vorgestellte Formular lässt sich für die Browserdarstellung angenehm stylen.

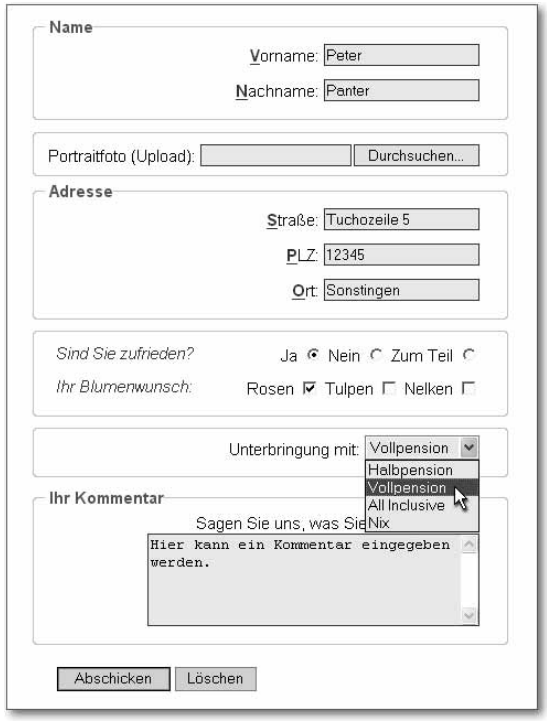

Abbildung 10.11 Das barrierefreie Formular

Hier der komplette Code des Formulars. Beachten Sie, dass wegen des Dateiumpoads das `enctype`-Attribut auf `"multpart/form-data"` und das `method`-Attribut auf `"post"` gesetzt werden muss:

```
<form action="/cgi/verarbeitung.pl" method="post"
 enctype="multipart/form-data">

<fieldset><legend>Name</legend>
   <label for="Vorname" tabindex="1" accesskey="v">
      <span>V</span>orname:
      <input type="text" name="Vorname" id="Vorname" />
   </label>
   <label for="Nachname" tabindex="2" accesskey="n">
      <span>N</span>achname:
      <input type="text" name="Nachname" id="Nachname"/>
   </label>
</fieldset>

<fieldset>
   <label for="Foto">Portraitfoto (Upload):
      <input type="file" name="Foto" id="Foto" />
   </label>
</fieldset>
<fieldset><legend>Adresse</legend>
   <label for="Str"><span>S</span>tra&szlig;e:
      <input type="text" name="Str" id="Str" accesskey="s"/>
   </label>
   <label for="PLZ"><span>P</span>LZ:
      <input type="text" name="PLZ" id="PLZ" accesskey="p"/>
   </label>
   <label for="Ort"><span>O</span>rt:
      <input type="text" name="Ort" id="Ort" accesskey="o"/>
   </label>
</fieldset>

<fieldset class="radiobox">
   <div><span>Sind Sie zufrieden?</span>
      <label for="rg1_r1">Ja
      <input type="radio" name="radgruppe1"
       id="rg1_r1" value="r1" checked="checked"/></label>
      <label for="rg1_r2">Nein
      <input type="radio" name="radgruppe1"
       id="rg1_r2" value="r2" /></label>
      <label for="rg1_r3">Zum Teil
      <input type="radio" name="radgruppe1"
      id="rg1_r3" value="r3" /></label>
   </div>
   <div><span>Ihr Blumenwunsch:</span>
      <label for="cg1_c1">Rosen
      <input type="checkbox" name="ckgruppe1"
```

```
       id="cg1_c1" value="c1" /></label>
      <label for="cg1_c2">Tulpen
      <input type="checkbox" name="ckgruppe1"
       id="cg1_c2" value="c2" /></label>
      <label for="cg1_c3">Nelken
      <input type="checkbox" name="ckgruppe1"
       id="cg1_c3"value="c3" /></label>
   </div>
</fieldset>

<fieldset>
   <label for="Verpflegung">
    Unterbringung mit:
   <select name="Verpflegung" id="Verpflegung"
    title="W&auml;hlen Sie die Art Ihrer Verpflegung">
      <option>Halbpension</option>
      <option>Vollpension</option>
      <option>All Inclusive</option>
      <option>Nix</option>
   </select>
   </label>
</fieldset>

<fieldset><legend>Ihr Kommentar</legend>
   <label for="Kommentar">
    Sagen Sie uns, was Sie von uns halten:<br/>
   <textarea rows="5" cols="35" name="Kommentar"
    id="Kommentar"></textarea>
   </label>
</fieldset>

<fieldset class="submitgroup">
   <input type="submit" value="Abschicken" />
   <input type="reset" value="L&ouml;schen" />
</fieldset>

</form>
```

Beispiel – CSS zur Gestaltung eines Formulars

Zur Erzielung der Darstellung in Abbildung 10.11 wird folgendes CSS verwendet, das hier erläutert werden soll.

Das Formular erhält eine Breite und wird zentriert:

```
form  { width:85%; margin:auto; }
```

Die Feldgruppen bekommen ein Margin, um sie auseinander zu rücken

```
fieldset {margin-bottom:10px; }
```

Alle Eingabefelder erhalten einen Rahmen und eine Hintergrundfarbe:

```
input {border:1px solid #555; background-color:#eee;}
```

Für Textareas und Selectboxen müssen die gleichen Eigenschaften definiert werden, wie für die restlichen Inputs, um sie optisch einander anzunähern.

```
textarea {border:1px solid #555; background-color:#eee;}
select {border:1px solid #555; background-color:#eee;}
```

Die `<label>`-Container werden auf `display:block` gesetzt. Sie übernehmen die Erzeugung der Darstellungsstruktur. Auf Zeilenumbrüche oder weitere Strukturelemente kann daher (fast überall) verzichtet werden. Das Label sorgt auch für die Ausrichtung der Inhalte und ein paar Margins:

```
label    { display:block; text-align:right;
           margin:5px 20px 10px 0; }
```

Für Hervorhebungen der Accesskeys ist ein `<span>`-Container da:

```
label  span           { text-decoration:underline;
                        font-weight:bold;color:#555;}
```

Die Input-Eigenschaften in Feldgruppen mit der Klasse `"radiobox"` werden geändert, um die allgemein definierten Rahmen für Checkboxen und Radiobuttons wieder zu entfernen und deren Hintergrundfarbe auf weiß zu setzen:

```
fieldset.radiobox input { border:0; background-color:#fff; }
```

Die Labels werden aus Platzgründen auf Inline gesetzt. Die Rechtsausrichtung der Inhalte und die Margins werden über den `<div>`-Container vorgenommen:

```
fieldset.radiobox label { display:inline; margin:0; }
fieldset.radiobox div { padding:0; text-align:right;
                        margin:10px 20px 10px 15px; }
```

Ein Text zur Gruppe wird nach links gefloatet:

```
fieldset.radiobox span { float:left;
                         font-style:italic; color:#555; }
```

Submit- und Resetbutton erhalten über die Feldgruppe der Klasse `"submitgroup"` eine dunklere Hintergrundfarbe:

```
fieldset.submitgroup input { border:1px solid #999;
                             background-color:#ddd; }
```

Hier die Farbe und Darstellung der Feldgruppenlegenden:

```
legend { color:#e20070; font-weight:bold; }
```

Nicht beeinflussbar ist leider die Darstellung des »Durchsuchen«-Buttons des Fileupload-Feldes.

10.2.7 Bilder, Scripte und Multimedia

Information wird in einer Website nicht nur über den Text, sondern auch über Bilder, Grafiken oder eingebundene Medien wie Videos oder Flashfilme übermittelt. Diese Information ist nur visuell ausgerichteten User Agents zugänglich – Suchmaschinen und Textbrowser, aber auch Screenreader bekommen hier Probleme. Gleiches gilt für Scripte, die den Informationsgehalt der Seite verändern, wie es beispielsweise bei AJAX-Anwendungen der Fall ist. Hier ist oft der Verzicht der einzige Weg zum Erzielen einer vollständigen Barrierefreiheit.

Eine eingefügte Grafik muss ein `alt`-Attribut erhalten und kann zusätzlich per `longdesc`-Attribut auf eine Beschreibungsseite verweisen (ob und wie dies unterstützt wird, ist Sache des User Agents):

```
<img src="diesel_05.gif" alt="Anteil Dieselfahrzeuge 2005"
 longdesc="diesel2005.html">
```

Alternativtexte für alle eingebundenen Medien

Ganz allgemein gilt immer: Stellen Sie für alle akustischen und visuellen Inhalte sowie für aktive Inhalte wie Applets und Scripts Textäquivalente zur Verfügung, damit die Informationen auch für User Agents zugänglich sind, die das entsprechende Medium nicht unterstützen (siehe BITV 1.1 bis 1.44).

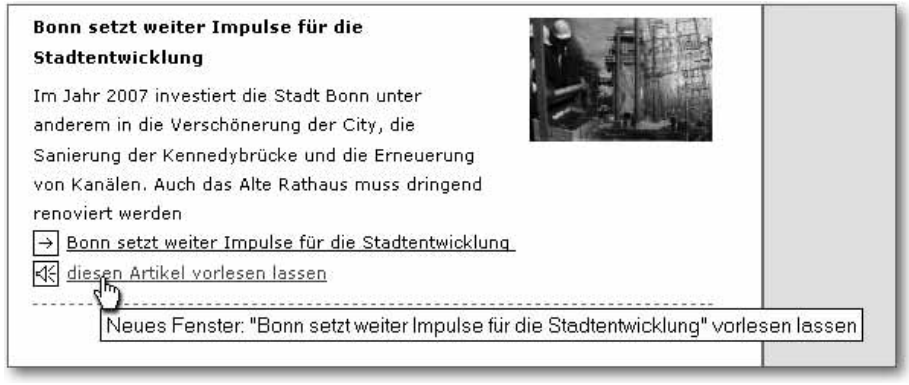

Abbildung 10.12 Artikel in Text- und Audioform (www.bonn.de)

Praxistipp

Auch hier hilft ein Textbrowser zur Kontrolle weiter. Sehen Sie anstelle von Grafiken oder Applets einen Alternativtext oder nur einen Dateinamen? Stehen für Videos oder Audiodateien Textäquivalente zur Verfügung? Gibt es einen `<noscript>`-Container als Stellvertreter für den Output eines Scriptes?

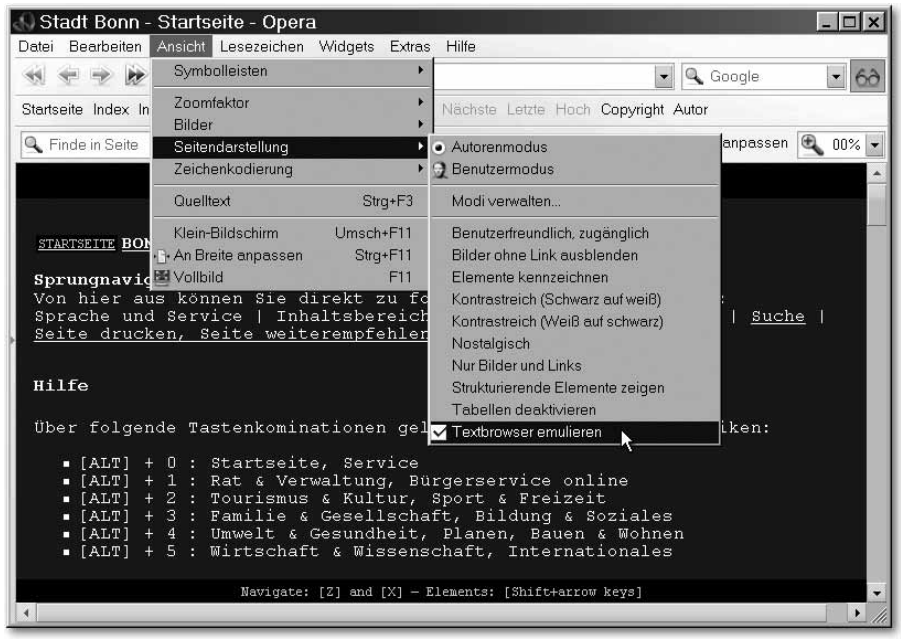

Abbildung 10.13 Textbrowser-Emulation in Opera 9.02

Kurzcheck zu Bildern, Grafikelementen und Diagrammen

Die folgenden Regeln gelten für das barrierefreie Einbinden von Bildern:

- Geben Sie für jedes `<img>` einen Ersatztext mit dem `alt`-Attribut an.
- Verwenden Sie `longdesc`, wenn ein Verweis auf eine genauere Beschreibung erforderlich ist, und stellen Sie diese zusätzlich über einen Link zur Verfügung (»D«-Link).
- Wenn Sie Grafiken für Links einsetzen, verwenden Sie das `alt`-Attribut zur Beschreibung des Linkziels.
- Setzen Sie, wenn möglich, keine Bilder für Texte oder Überschriften ein und stellen Sie ansonsten ein adäquates Textäquivalent zur Verfügung.
- Wenn es möglich ist, zur Darstellung von Inhalten anstelle von Grafiken spezielles Markup (z. B. MathML) einzusetzen, tun Sie dies.

Abbildung 10.14 Linktitel-Tooltip, alt-Text als Bildalternative (www.bonn.de)

Kurzcheck zu Imagemaps

Falls Sie mit Imagemaps arbeiten wollen oder müssen, beachten Sie Folgendes:

- Verwenden Sie nur clientseitige Imagemaps.
- Sollten Sie doch serverseitige Imagemaps einsetzen müssen, stellen Sie zusätzlich Textlinks als Alternative zur Verfügung.
- Geben Sie mit dem `alt`-Attribut einen Ersatztext für das Gesamtbild an.
- Machen Sie Linkziele über das `alt`-Attribut des `<area>`-Elements kenntlich.
- Stellen Sie eine Liste über die Links einer Seit zur Verfügung.

Kurzcheck zu Multimedia und Applets

Leicht abweichende Regeln gelten für die Einbindung von Multimedia, wobei es im Prinzip gleichgültig ist, ob dies per `<applet>`-Element (eher schlecht) oder schulmäßig per `<object>`-Container erfolgt. Einige nicht unberechtigte Forderungen wie das generelle Untertiteln von Videos (womöglich mehrsprachig) sind aus Kostengründen leider eher Wunschdenken.

- Stellen Sie für Videos und Animationen alternative Textbeschreibungen zur Verfügung. Bieten Sie diese notfalls auf einer separaten Seite an, wenn es der Umfang erfordert.
- Machen Sie die Alternativinformationen über Tastatur zugänglich (also nicht etwa per Klick auf die Darstellungsfläche des Mediums).
- Stellen Sie für akustische Inhalte von Videos Untertitel zur Verfügung.
- Vermeiden Sie Blink- und Flackereffekte über den gesamten Bildschirmbereich (diese können epileptische Anfälle auslösen).

- Achten Sie bei Applets möglichst auf unmittelbare Barrierefreiheit und stellen Sie Textalternativen für nicht barrierefreie Applets zur Verfügung.
- Stellen Sie Audio-Inhalte zusätzlich in Textform zur Verfügung.
- Stellen Sie im Zweifelsfall eine alternative, barrierefrei zugängliche Seite ohne Multimediainhalte zur Verfügung.

Kurzcheck zu Scripten

Der wichtigste Punkt bezüglich des Einsatzes von Scripten, sofern diese durch den Nutzer gesteuert werden sollen, betrifft die Verwendung geräteunabhängiger Eventhandler. Problematisch sind auch Scripte, die den Inhalt einer Seite verändern – dies kann bereits ein Bildaustausch sein, für den auch den Austausch des betreffenden Erklärungstextes nicht vergessen werden sollte.

- Verwenden Sie, wo möglich, logische, von Eingabegeräten unabhängige Eventhandler (z. B. `onfocus` statt `onclick` oder `onkeydown`).
- Achten Sie, wenn möglich, bei Scripten auf unmittelbare Barrierefreiheit (z. B. nicht ausschließlich visuelle Rückmeldung über den Zustand).
- Stellen Sie für nicht barrierefreie Skripte `<noscript>`-Alternativen bereit.

10.2.8 Weiterleitungen, Popups, dynamische Inhalte

Das Öffnen von Links in neuen Fenstern oder automatische Popup-Fenster sind für verschiedene User Agents ein Problem und sollten vermieden werden. Bei Weiterleitungen und Meta-Refresh beruht die Problematik eher darin, dass diese durch den Nutzer schwer oder gar nicht steuerbar ist. (Hinweis: Die Webdeveloper-Toolbar ermöglicht es, Weiterleitungen zu Testzwecken zu deaktivieren.)

Kurzcheck zu Weiterleitungen

- Verzichten Sie auf automatische Weiterleitungen oder periodischen Neuaufruf der Seite durch Meta-Refresh.
- Wenn ein automatischer Neuaufruf vorgenommen werden muss, machen Sie den Nutzer darauf aufmerksam.
- Wenn eine Weiterleitung erforderlich ist, stellen Sie hierfür einen normalen Textlink zur Verfügung oder steuern Sie die Weiterleitung serverseitig.

Kurzcheck zu Popup-Fenstern

Popups stellen eine Barriere dar, wenn sie technisch nicht darstellbar sind (der Nutzer erhält die Information gar nicht), oder den Fokus auf eine Weise auf sich ziehen, dass es den Nutzer verwirrt (bei manchen Mobilgeräten). Also:

- Stellen Sie zumindest keine relevanten Inhalte in Popup-Fenstern dar und, sofern möglich, verzichten Sie vollständig auf das Öffnen neuer Fenster.
- Informieren Sie den Nutzer, wenn sich ein Link in einem neuen Fenster öffnet.

Abbildung 10.15 Ankündigen eines neuen Fensters (www.bonn.de)

Kurzcheck zu dynamischen Inhalten

Schwer zu verwirklichen ist folgende Forderung, wenn Sie externe Inhalte wie Börsenticker oder Wetterprognosen in Ihre Site einbinden. Sofern dies über IFrames geschieht, haben Sie damit jedoch bereits eine weitere Hürde.

- Sorgen Sie dafür, dass sich Textalternativen zu dynamischen Inhalten parallel zu diesen aktualisieren.

Kurzcheck zu Download-Dateien

Ein Downloadlink sollt von einem normalen Link auf den ersten Blick unterscheidbar sein, damit der Nutzer nicht versehentlich einen Download auslöst. Dies kann durch ein Link-Icon geschehen (siehe Abbildung 10.16). Art und Größe der Datei sollten deutlich gemacht werden. Der Dateiname sollte kurz, aber beschreibend sein, um einer möglichen Verstümmelung beim Speichern vorzubeugen und die Datei für den Nutzer auffindbar zu machen (geben Sie also auch den Dateinamen vor dem Download bekannt).

Abbildung 10.16 Download-Links für PDF-Dateien

- Kennzeichnen Sie PDF-Downloads deutlich als solche.
- Geben Sie bei PDF-Downloads einen Link zum Adobe Acrobat-Reader an.

- Bieten Sie unterschiedliche Formate an (.pdf, .doc, .rtf, .txt).
- Beschränken Sie sich bei den Dateinamen von Downloaddateien auf die MS-DOS-Konvention.

10.2.9 Barrierefreie Inhalte

Auch Ihre Inhalte selbst beziehungsweise die Art ihrer Vermittlung sollten so barrierefrei wie möglich sein – selbstverständlich gibt es Grenzfälle, wo weder Abstriche an der Darstellung oder Stilistik gemacht werden können oder eine weitere Vereinfachung ohne unzulässige Verfälschung möglich ist. Dennoch bringt die Überprüfung der Texte einer Website Vorteile: Je verständlicher Sie Ihre Informationen formuliert haben, desto mehr kommt beim Nutzer an.

Kurzcheck zur Verständlichkeit der Inhalte

- Strukturieren Sie die Seiten klar und übersichtlich.
- Bilden Sie übersichtliche, kurze Informationsblöcke.
- Verwenden Sie für ihre Inhalte einfache und verständliche Sprache.
- Prüfen Sie sorgfältig auf Rechtschreibung und setzen Sie gegebenenfalls professionelle Korrektoren ein, um die Textqualität zu gewährleisten.
- Erläutern Sie auftretende Fachausdrücke gegebenenfalls in einem Glossar.
- Stellen Sie Inhalte auch in anderen Sprachen zur Verfügung, seien Sie dabei aber möglichst konsistent.

10.3 Zusammenfassung

Aus diesem Kapitel sollten Sie folgende Aussagen mitnehmen:

- Barrierefreiheit ist kein Aspekt, der nur Randgruppen betrifft, sondern ein Muss für zeitgemäßes Webdesign.
- Barrierefreiheit bezieht sich nicht nur auf bestimmte Nutzergruppen, sondern auch auf Typen von User Agents (z. B. Suchmaschinen, Screenreader).
- Barrierefreiheit ist bei konsequentem und korrektem Einsatz von (X)HTML und CSS relativ einfach zu erreichen.
- Barrierefreiheit betrifft die Zugänglichkeit der Inhalte. Mit einer barrierefreien Navigation steht und fällt der Erfolg einer Website.
- Sofern Inhalte in bestimmter medialer Form vorliegen (Bild, Video, Audio), sollten Alternativinhalte in Textform bereitgestellt werden.

Wie geht es weiter?

Das folgende Kapitel 11, *Werkzeuge für Webdesigner*, gibt Ihnen eine Übersicht über Tools und Programme, die für Sie als Webentwickler von Nutzen sein können. Themen sind nicht nur die Hilfsmittel zur Erstellung barrierefreier Sites, wie sie in diesem Kapitel angesprochen wurden, sondern auch Editoren und Browser-Plug-ins, die den Designeralltag erleichtern.

In diesem Kapitel finden Sie – ohne Anspruch auf Vollständigkeit – eine kleine Übersicht über die Werkzeuge des Webdesigners, angefangen beim Editor bis zu Hilfsprogrammen und Browsertools zur Erleichterung des Alltags.

11 Werkzeuge für Webdesigner

Ohne Editoren geht bei der Erstellung von Webseiten nichts – wobei die Grundanforderungen gering sind, denn immerhin handelt es sich bei HTML und CSS um reine Textformate, die dementsprechend mit jedem beliebigen Texteditor bearbeitet werden können.

11.1 Editoren

Beliebter Einstiegseditor ist wegen der oben erwähnten geringen Anforderungen das Notepad von Windows oder ein vergleichbarer Editor, wie er in jedem Betriebssystem vorzufinden ist. Es gibt sogar Vorteile in der Simplizität – sofern der Autor darauf geachtet hat, eine neu erstellte Datei im richtigen Format abzuspeichern, hat er alle anschließend auftretenden Fehler selbst zu verantworten: Das Programm leistet zwar keinerlei Hilfestellung, greift aber auch nie unerwünscht ins Geschehen ein (Stichwort »Code Completion«, also die automatische, nicht immer treffsichere Ergänzung eingegebener Code-Fragmente).

11.1.1 Quelltexteditoren

Früher oder später ist ein wenig mehr Komfort gewünscht – zumindest ein Zeilenzähler sollte her und eventuell auch Syntax-Highlighting zur Verbesserung der Übersicht. Will man dicht an der Programmierbasis bleiben und den Code ständig direkt vor Augen haben, bietet sich ein Quelltexteditor an.

Sicherlich keinen Fehler begeht man, wenn man zu bewährten Programmen greift wie die universellen Texteditoren **Ultra Edit** (Windows; **www.ultraedit.com**), **SciTE** (Windows, Linux; **www.scintilla.org**) oder **jEdit** (Linux, MacOS, Windows; **www.jedit.org**).

Alternativen unter Windows, wenn auch geringfügig spartanischer ausgestattet, bieten das sehr solide **Notepad++** (Windows), der **Crimson Editor** (Windows) oder der Shareware-Editor **Edit4Win** (Windows) von Mirko Böer.

Eine Stufe darüber stehen die ganz auf HTML spezialisierten Texteditoren. Für Windows wären hier – neben vielen anderen – die Freeware-Editoren **Arachnophilia** von Paul Lutus und **Weaverslave**[1] von Thomas Weinert, **Phase 5** von Hans-Dieter Berretz und **Scribe** von Jan Uhlenbrok zu nennen, sowie als mächtige, aber kommerzielle Alternative, Macromedias (bzw. inzwischen Adobes) **Homesite**.

Dünner gesät ist der Markt für Mac OS. Hier gibt es den Open-Source-Editor **Bluefish** (Mac/Linux) und den kommerziellen, aber preiswerte **PageSpinner** (Mac OS).

Für Linux/Unix sind die komfortablen Open-Source-Editoren **Quanta+** (Linux) und **Screem** (Linux, BSD Unix) zu empfehlen.

Crimson Editor

Ein sehr schöner, kompakter Freeware-Editor ist Ingyu Kangs **Crimson Editor** (Abbildung 11.1), der sich hervorragend als Notepad-Ersatz eignet und alle benötigten Features besitzt wie Unicodefähigkeit, Multi Undo/Redo, Syntax-Highlighting (für verschiedene Sprachen, erweiterbar), Zeilenanzeige und Tabs zum gleichzeitigen Bearbeiten mehrerer Dateien, dabei jedoch simpel und gradlinig in der Bedienung ist. Bei genauerem Hinsehen existieren aber auch fortgeschrittene oder einfach nur praktische Features wie ein eingebauter FTP-Client, Projektverwaltung oder Makroerzeugung zur Automatisierung.

Anmerkung: Syntax-Highlighting-Erweiterungen in zahlreichen Sprachen von Cobol über Lisp bis TypoScript sind auf der Website des Autors zu beziehen.

Plattform: Windows (ab 95 oder NT)

Status: Freeware (Spenden sind dem Autor willkommen)

Beurteilung: Ebenso professionelle Alternative zu Ultra Edit

Downloadgröße/Festplattenbedarf: 1,2 MB/4 MB

Website: www.crimsoneditor.com

1 Ehemals »HTMLPad«.

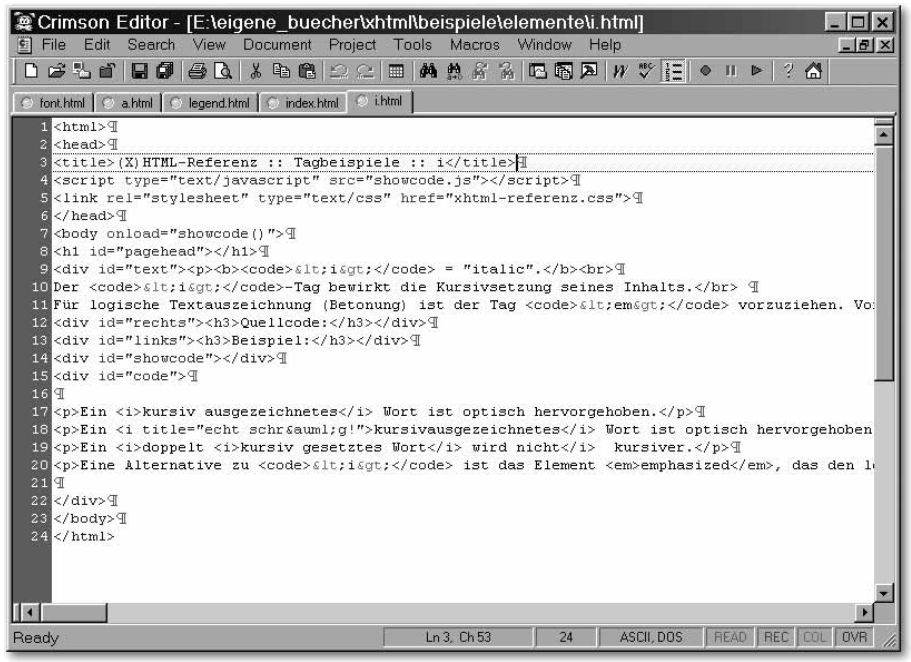

Abbildung 11.1 Crimson Editor mit mehreren geöffneten Dokumenten

Phase 5 HTML-Editor

Eindeutiger auf HTML und Webanwendungen festgelegt ist **Phase 5** (Abbildung 11.2) von Hans-Dieter Berretz. Die für Privatanwender und Schulen freie Software (für professionelle Nutzer wird eine sehr faire Lizenzgebühr von derzeit 30 Euro erhoben) kann mit kommerziellen professionellen Umgebungen sehr gut mithalten, ohne deren überbordenden Funktionen-Overkill und entsprechenden Ressourcenhunger mitzubringen – auf der Festplatte benötigt das Programm weniger als 3 MB. Als interne Browservorschau wird Internet Explorer verwendet. Interessant: Phase 5 unterstützt, was nicht allgemein üblich ist, auch VBScript.

Plattform: Windows 95, 98, ME, NT, 2000, XP

Status: Freeware für Privatanwender/Schulen; sonst Lizenz erforderlich

Beurteilung: Sehr schöner HTML-Editor, wenn auch nicht ohne Lernkurve

Downloadgröße/Festplattenbedarf: 2,3 MB/2,3 MB

Website: www.ftp-uploader.de

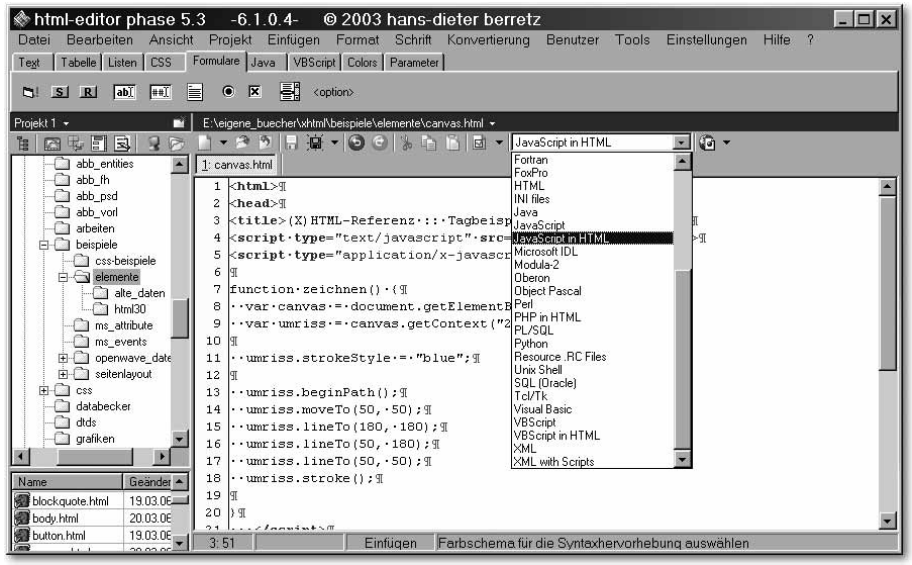

Abbildung 11.2 Phase 5 HTML-Editor

11.1.2 WYSIWYG HTML-Editoren

Will man nicht oder nicht nur mit dem Dokumentquelltext arbeiten, sondern auch direkt in der Ansicht der Webseite editieren, so hat man die Auswahl zwischen zahlreichen Lösungen in Form so genannter WYSIWYG-Editoren. Jedoch ist das »What you see is what you get« mit Vorsicht zu genießen – gerade wenn es um CSS-basierte Seitenlayouts geht, versagen einige dieser Editoren, und man muss wie gewohnt mit Browservorschauen arbeiten. Auch das Versprechen, komplett ohne HTML-Kenntnisse auszukommen, wird in der Regel nicht eingelöst und ist bei den professionelleren Programmen dieser Gattung ohnehin von wenig echtem Interesse.

Die Mehrzahl der Editoren dieser Kategorie ist auf die Windows-Plattform ausgerichtet, für die es in Form des **Namo WebEditor** (**www.namo.com**) oder von **SuperHTML** (**www.superhtml.de**) auch preiswertere Shareware-Editoren gibt.

Die meisten professionellen Webdesigner entscheiden sich zwischen den beiden »Großen«, womit **Dreamweaver** und **GoLive** (das von Adobe vermutlich eingestellt werden wird) gemeint sind, die neben ihrer Windows-Versionen beide auch als Mac OS-Programme vorliegen. Mit **Nvu** taucht ein Konkurrent, der derzeit vom Funktionsumfang noch nicht ganz mithalten kann, jedoch auch auf dem Open-Source-Sektor und damit auch für Linux-Plattformen auf.

Nvu

Nvu (Abbildung 11.3) ist ein verhältnismäßig neuer, auch mit deutscher Oberfläche vorliegender WYSIWYG-Editor aus der Open-Source-Gemeinde, der auf der Mozilla-Plattform und deren Layout-Engine Gecko basiert. Neben einer Version für Windows steht er auch für Linux und Mac OS zur Verfügung (weitere Plattformen in Vorbereitung). Die HTML-Vorschau ist, wie Gecko verspricht, dementsprechend makellos.

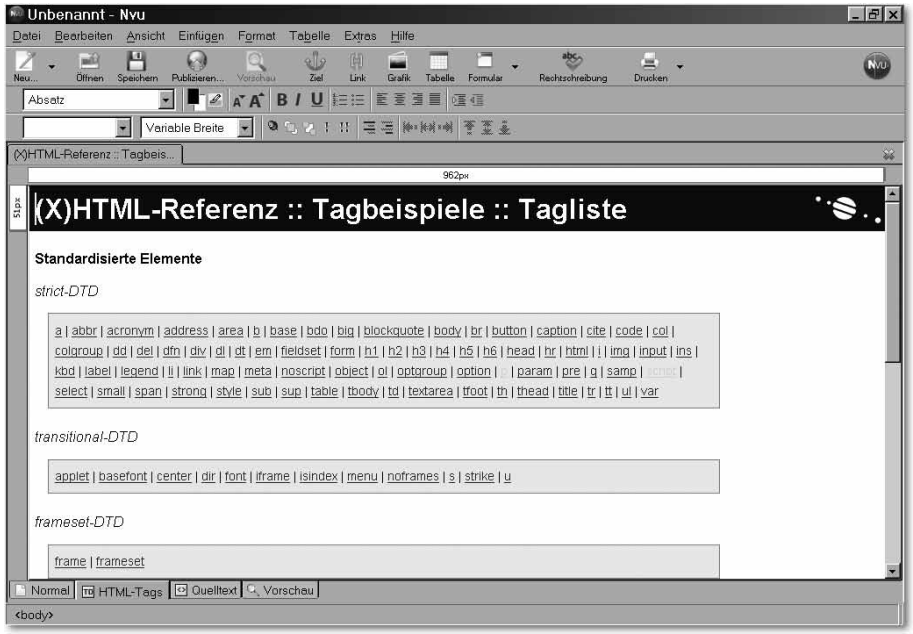

Abbildung 11.3 Nvu (Version für Windows)

Von der Anzahl der Features kann Nvu zwar derzeit noch nicht mit der kommerziellen Konkurrenz mithalten, zeichnet sich dafür jedoch durch eine übersichtliche Bedienoberfläche und geringen Ressourcenbedarf aus (das installierte Programm belegt gerade knapp über 20 MB auf der Festplatte. Dennoch unterstützt Nvu die Erstellung von Webanwendungen mit HTML, XHTML, CSS, JavaScript und XML. Für CSS ist der Editor »CaScadeS« bereits integriert, als Unterstützung zur Script-Programmierung kann Robert Gindas JavaScript-Debugger »Venkman« eingesetzt werden, der auch in einer von Thorsten Fritz übersetzen Version vorliegt.

Höchst interessant ist die generelle Erweiterbarkeit von Nvu mit Erweiterungen aus dem Mozilla/Firefox Umkreis (dazu zählt Venkman), die das Programm in naher Zukunft noch mächtiger (und individualisierbar) zu machen verspricht.

[»] *Hinweis:* In Mozilla oder Firefox kann Nvu über das Kontextmenü so eingebunden werden, dass die aktuell gezeigte Webseite unmittelbar im Editor geöffnet werden kann. Hierzu benötigt man die Erweiterung »Launchy« (von Henrik Gemal; **http://gemal.dk/mozilla/launchy.html**, launchy.xpi).

Plattform: Windows, Linux (verschiedene), Mac OS X, OS/2 (in Vorbereitung)

Status: Open-Source-Software

Beurteilung: Open-Source-Konkurrenz der kommerziellen Pendants

Downloadgröße/Festplattenbedarf: ca. 6 MB/21 MB (je nach Plattform)

Website: www.nvu-composer.de

11.2 Toolbars und weitere nützliche Werkzeuge

11.2.1 Developer Toolbars

Unter dem Oberbegriff »Developer Toolbars« sind solche Browsererweiterungen zusammengefasst, die allgemein nützliche Lösungen für die Alltagsnöte eines Webentwicklers zur Verfügung stellen.

Mozilla Web Developer-Toolbar

Für Webdesigner eigentlich unentbehrlich ist die **Mozilla Web Developer Toolbar** (auch für Firefox, Flock, Seamonkey; siehe Abbildung 11.4), die von Chris Pederick entwickelt und als Freeware zur Verfügung gestellt wird. Die Toolbar ist auf allen Plattformen einsetzbar, die von den genannten Browsern unterstützt werden, also Windows, Linux, Mac OS X. Für die ältere Version 1.0.2 ist zudem eine deutsche Fassung erhältlich (beschrieben wird die aktuellere englischsprachige Originalversion 1.1.3).

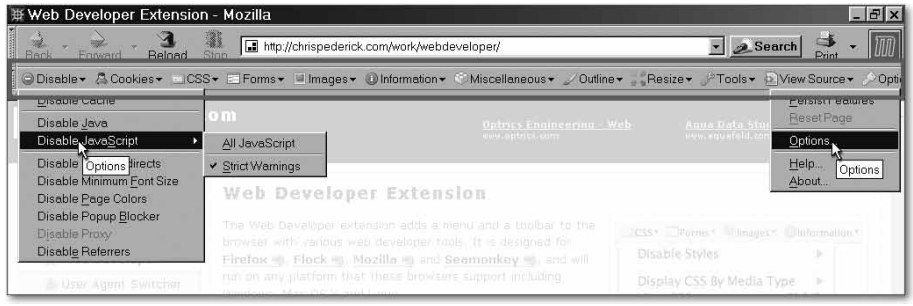

Abbildung 11.4 Web Developer Toolbar (Firefox, Mozilla, Flock, Seamonkey)

Die Toolbar bietet mit Hilfe mehrerer Menügruppen Möglichkeiten zur Steuerung der Darstellung sowie Struktureinblicke in das aktuell im Browser dargestellte Dokument.

Disable
Deaktivieren von JavaScript, Java, Browsercache, Popup-Blocker oder Farbdefinitionen (Schrift- und Hintergrundfarben).

Cookies
Gezieltes Deaktivieren, Löschen, Setzen und Betrachten von Cookies.

CSS
Gezieltes Aktivieren oder Deaktivieren von Styles, Forcieren von User Styles oder Aktivieren medienspezifischer Styles (Print, Handheld).

Hinweis: Das Feature **View Styleinformation** offenbart durch Klick auf ein Element die Summe dessen CSS-Formatierungen. **[«]**

Forms
Ermöglicht die (als nicht ganz unproblematisch zu wertende) Manipulation von Formularen – von Autocomplete bis zu Entfernen von `maxlength`-Properties oder dem Umstellen von `GET` nach `POST` und zurück.

Images
Ermöglicht das pauschale oder gezielte Abschalten von Bildern, das Outlining nach verschiedenen Kriterien (kein oder leeres `alt`-Attribut, skaliertes Bild) und die Ausgabe von Bilddaten (Pfad, Dateigröße, Abmessungen).

Information
Sichtbarmachung von Metatag-Inhalten, Accesskeys, Tabindex, Ankerpunkten, `id`- und `class`-Attributen, `title`-Attributen und weiteren Informationen über die Seite bis zu einer Liste der verwendeten Farbwerte.

Hinweis: Der Menüpunkt **View Javascript** zeigt sämtliche internen oder verlinkten Scriptdaten. **[«]**

Miscellaneous (Verschiedenes)
Eine Reihe praktischer Tools und Features vom Screenlineal mit Positions- und Abmessungsangaben, dem Einblenden eines Gitternetzes bis hin zur Option, Kommentare und versteckte Elemente einzublenden oder der Simulation einer Handheld-Anzeige (Small Screen Rendering). Neu in der Version 1.1.3 ist die Option **Small Screen Rendering**, die Mozilla in dieser Beziehung mit Opera gleichziehen lässt.

Outline
Ermöglicht das Outlining beliebiger Elemente (Tabellen, Div-Container, Images, Überschriften), wobei auch gezielt positionierte Elemente (mit Berücksichtigung der Art der Positionierung) hervorgehoben werden können.

Resize
Ermöglicht die Größenveränderung des Browserfensters (voreingestellt 800 x 600), bietet Ausgabe der aktuellen Fenstergröße und eine Zoomfunktion.

Interessant: Über den Hauptmenüpunkt **Options > Resize** können beliebige weitere, an dieser Stelle abrufbare Fensterzielgrößen hinzugefügt werden.

Tools
Links zu Validierungsmöglichkeiten (HTML, CSS, Accessibility), Test der Ladegeschwindigkeit (über **www.websiteoptimization.com**; nicht für lokale Files), ein DOM-Inspektor und Zugriff auf Java- und JavaScript-Konsole.

View Source
Button zum Direktzugriff auf die Quelltextanzeige des Browsers.

Options
Unter anderem Zugriffsmöglichkeit auf die Toolbar-Parameter, worunter neben den Keyboardshortcuts die Konfiguration der über **Tools** eingebundenen Validatoren oder (s. o.) der über **Resize** ansprechbaren Fenstergrößen zählt.

- Download/Information: **www.chrispederick.com/work/webdeveloper/**

Internet Explorer Developer Toolbar

Direkt von Microsoft stammt die dementsprechend nur unter Windows im Internet Explorer einsetzbare **Internet Explorer Developer Toolbar** (ab IE 6.0, inzwischen auch für Internet Explorer 7 verfügbar). Aktuell ist die Version **Beta 3** vom 12. Januar 2007, die funktional gegenüber der Toolbar von Chris Pederick stark aufgeholt hat. Im Gegensatz zur Vorgängerversion kann man von einer permanent sichtbaren »Toolbar« eigentlich nicht mehr sprechen – die Funktion versteckt sich hinter einem unauffälligen Button in der Schaltflächenleiste (siehe Hervorhebung in Abbildung 11.5).

Diese blendet das eigentliche Tool ein (hier als »DOM Explorer« bezeichnet), der zunächst das untere Drittel des Browserfensters einnimmt, jedoch (über »unpin«) auch als separates Fenster genutzt werden kann. Die Funktionen des Developer-Tools sind über dessen Menü und teilweise über Icons erreichbar.

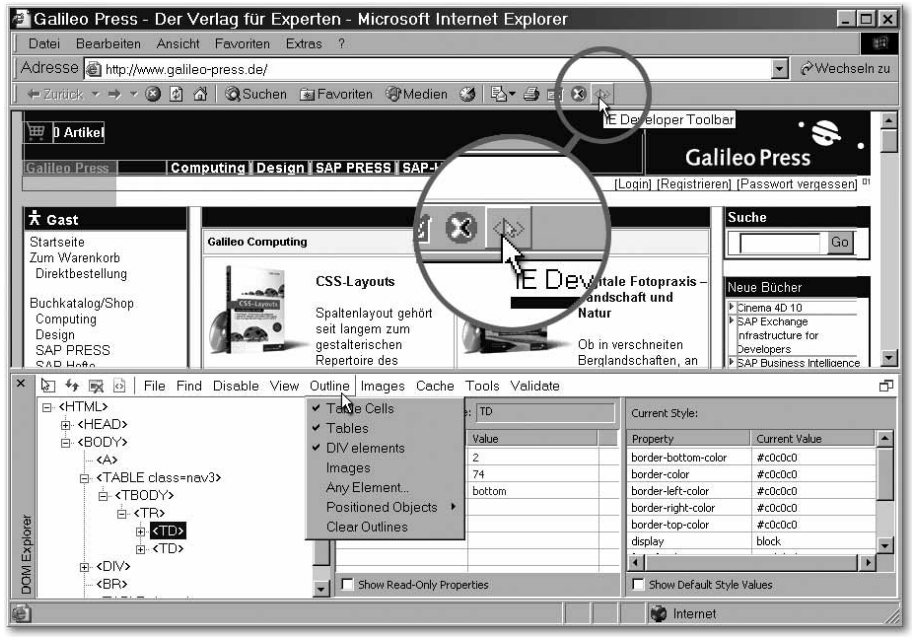

Abbildung 11.5 MS Developer Toolbar für Internet Explorer

File
Ermöglicht über Undo das Löschen der Hervorhebungen im Dokument

Find
Ermöglicht das Anwählen eines Elements per Klick in das Dokument (diese Funktion ist auch per Icon zugänglich) oder über eine Eigenschafts-Suchfunktion und analysiert es im DOM Explorer. Das aktuell betrachtete Element wird über ein Outline hervorgehoben.

Disable
Ermöglicht das Deaktivieren des Browsercaches, von Bildern, Scripten (pauschal), Cookies, CSS oder des Popup-Blockers.

View
Zeigt Accesskey- und Tabindex-Definitionen, Linkpfade sowie `id`- und `class`-Attributangaben.

Outline
Markiert mittels farbiger Outlines Tabellen, Tabellenzellen, `div`-Container, positionierte Elemente (nach Art der Positionierung) oder (frei konfigurierbar) beliebige, durch den User festzulegende Elemente.

Validate
Validiert lokal oder remote HTML, CSS, Accessibility (WAI, 508), Links oder Feeds.

[»] *Hinweis:* Mehrere Validierungsanfragen lassen sich zu einem Set zusammenfassen und »en bloc« ausführen.

Images
Gibt Informationen über Bilder, von Abmessungen über Dateigröße, Pfade und Wert des `alt`-Attributs zu einem »Image Report«, der in einem neuen Fenster alle Bilder eines Dokuments mit den relevanten Informationen auflistet.

Resize
Bietet die Möglichkeit, die Größe des Browserfensters auf eine der vorgegebenen Abmessungen zu bringen. Es können beliebige Werte hinzugefügt werden.

Misc
Unter »Verschiedenes« findet man neben Möglichkeiten der Cache- und Cookieverwaltung und direkten Links zu verschiedenen W3C-Spezifikationen auch einen Color Picker, der mittels Pipette beliebig Farbinformationen aufnehmen und deren Wert in der Zwischenablage platzieren kann, dabei aber auf den Bereich des Browserfensters beschränkt ist.

Show Ruler
Blendet eine Linealfunktion ein, die in beliebigen Winkeln und Richtungen die Distanz zwischen zwei Punkten ausmessen kann. Das Lineal kann an Elementen einrasten und auf horizontale bzw. vertikale Richtung beschränkt werden.

- Download: **www.microsoft.com/downloads/** (Rubrik: Internet)

Web Accessibility Toolbar

Derzeit ebenfalls nur für den Internet Explorer verfügbar ist die **Web Accessibility Toolbar** des australischen National Information and Library Service (NILS, **www.nils.org.au** – mittlerweile aufgegangen in »Vision Australia«, **www.visionaustralia.org.au**), die auch in einer eingedeutschten Version von »Web for all« (**www.webforall.info**) erhältlich ist.

Die Toolbar (siehe Abbildung 11.6) bietet eine Reihe Funktionen, die den Developer-Toolbars entsprechen, darüber hinaus jedoch weitere Validierungsmöglichkeiten nach Accessibility-Gesichtspunkten und interessante wie erhellende Simulationen (Umsetzung des Screens als Grausstufendarstellung, Fehlsichtigkeitssimulationen), Tools (Kontrastanalyse, Textbrowser) und schließlich Linklisten auf einschlägige Bestimmungen und Richtlinien.

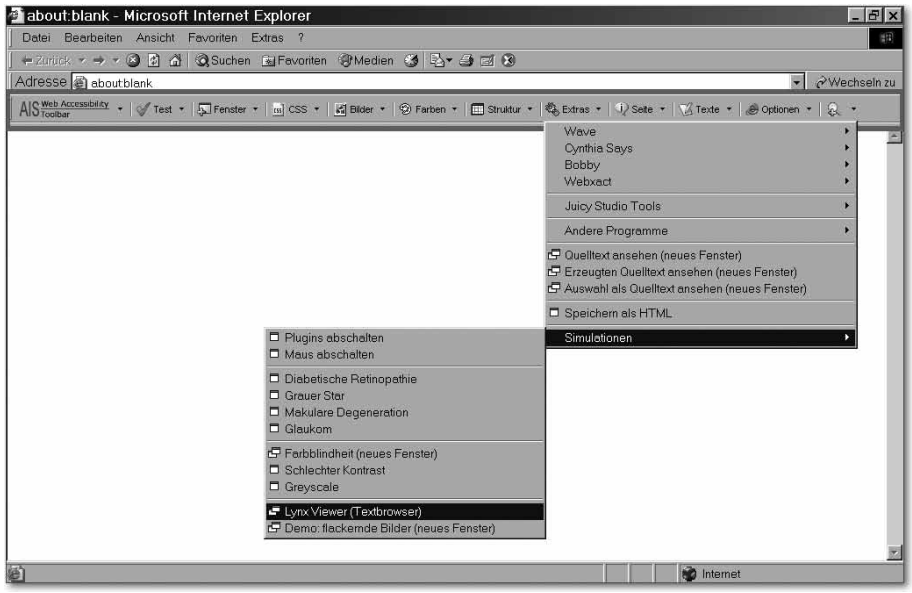

Abbildung 11.6 Accessibility Toolbar, Menü »Extras« (nur Internet Explorer)

Plattform: Windows 98, NT, 2000, XP; Internet Explorer ab 5.0

Download: www.webforall.info/html/deutsch/aistoolbar.php

11.2.2 Wave Toolbar

In Form der **Wave-Toolbar 1.0** (Abbildung 11.7) steht auch den Mozilla/Firefox-Nutzern eine kostenlose Accessibility-Toolbar zur Verfügung, die sich allerdings auf die Nutzung des Wave-Portals (**www.wave.webaim.org**) und dessen Wave 3.0 Accessibility Tool zur Beurteilung von Webseiten beschränkt.

Die Verwendung ist im Gegenzug einfach – es kann entweder die aktuell gezeigte Seite geprüft werden (das Dokument muss online erreichbar sein) oder eine beliebige Webadresse in das URL-Feld der Toolbar eingegeben werden.

Da das von Wave gezeigte Prüfergebnis (alle Darstellungsoptionen sind von Haus aus gleichzeitig aktiv) zunächst sehr unübersichtlich wirkt, ist es anzuraten, eigene Präferenzen einzustellen, die per Cookie lokal gespeichert und jederzeit geändert werden können. Hierzu stellt Wave eine gut gegliederte Konfigurationsseite zur Verfügung.

Browser: Internet Explorer, Mozilla, Firefox, Netscape

Download: www.wave.webaim.org

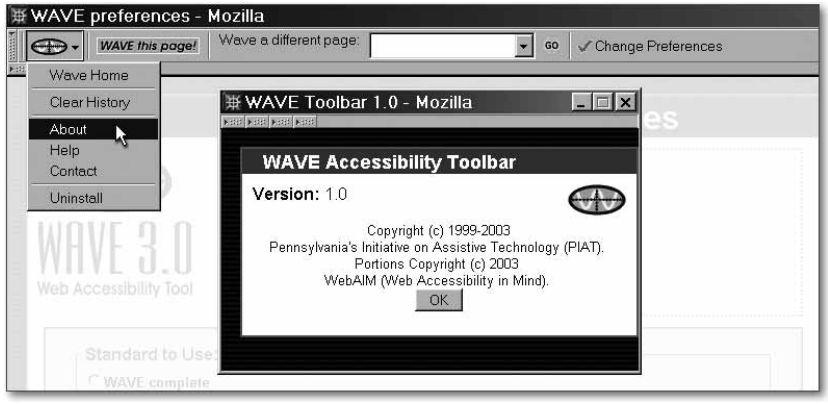

Abbildung 11.7 Die Wave Toolbar in ihrer Version 1.0 (Mozilla)

11.2.3 Accessibility-Tools

Farbkontrast-Analyzer 1.0

Ebenfalls vom australischen National Information and Library Service erhältlich ist der **Color Contrast Analyzer**, ein Tool zur Kontrastanalyse, das die Lesbarkeit einer Webseite nach Schrift- und Hintergrundfarbe beurteilt. Beide Farbwerte werden mit einer Pipette (mit Lupenfunktion) aus der Seite aufgenommen, anschließend erfolgt eine automatische Bewertung. Bei »Web for all« ist auch hier eine eingedeutschte Version erhältlich (siehe Abbildung 11.8).

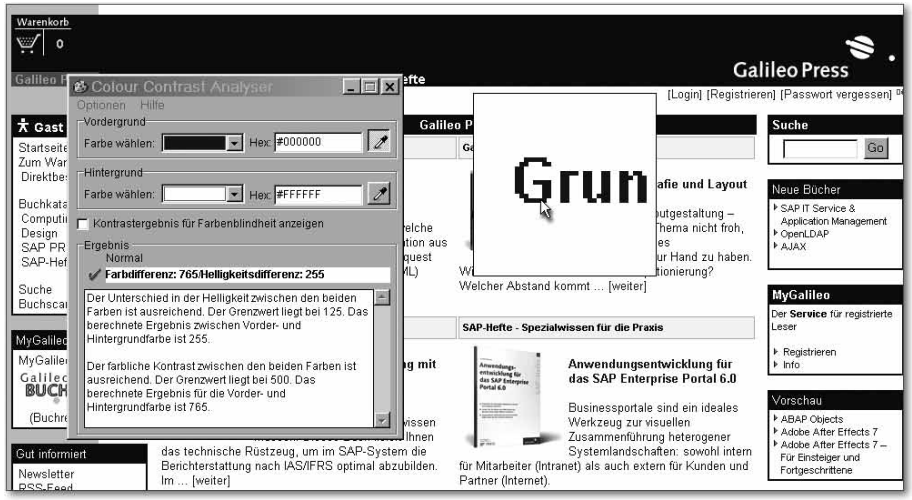

Abbildung 11.8 Farbkontrast Analyzer bei Hintergrundfarbwahl

Download: www.webforall.info/html/deutsch/col_analy.php

11.2.4 Bildschirmlineale

Measure It (Firefox)

Ein gutes Bildschirmlineal als Firefox-Erweiterung stellt **Measure It** von Kevin Freitas dar (Abbildung 11.9). Das Tool ist über ein Icon unten links in der Browserstatusleiste zu aktivieren. Anschließend kann ein rechteckiger Messbereich beliebig über die Seite gelegt werden, deren Position und Abmessungen ausgegeben werden. Die Darstellung ist optisch angenehmer als die des Screenrulers der IE-Toolbar, bietet jedoch keine Möglichkeit des »Einrastens« an Elementen.

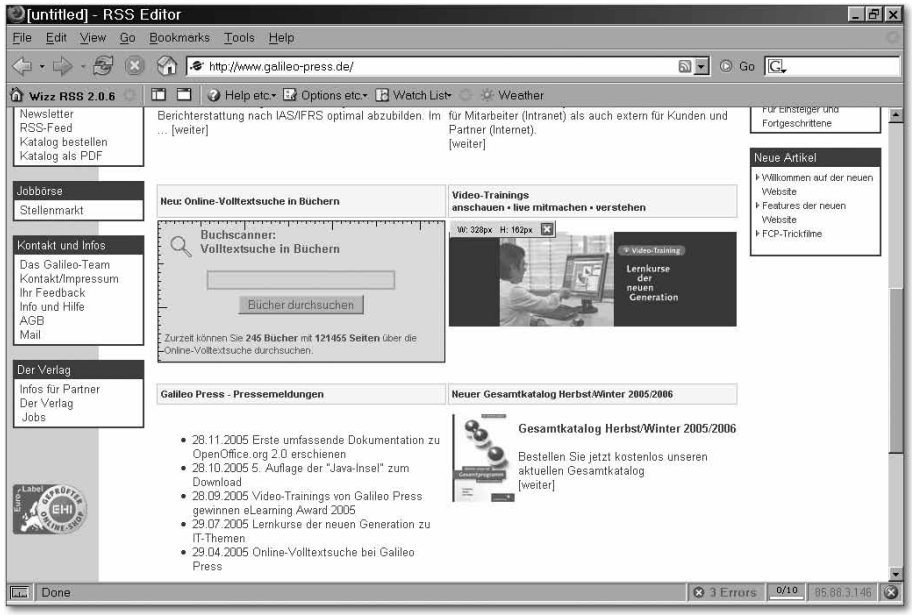

Abbildung 11.9 Measure It bei der Arbeit

Anmerkung: Wer die Webdeveloper Toolbar installiert hat, benötigt Measure It eigentlich kaum, da die dort unter »Verschiedenes« angebotene Lösung **Display Ruler** quasi funktionsgleich ist.

Plattform: Firefox

Status: Freeware-Extension

Beurteilung: gutes, browserinternes Bildschirmlineal

Downloadgröße/Festplattenbedarf: 19 KB (xpi-Datei)

Website: www.kevinfreitas.net

11.2.5 Colorpicker und Farbtools

ColorZilla 0.8.2

Für Firefox bietet die Erweiterung **ColorZilla** (Iosart, Abbildung 11.10) die Möglichkeit, mittels einer Farbpipette eine beliebige Farbe innerhalb einer im Browser gezeigten Webseite aufzunehmen. Das Tool ist über ein Icon unten links in der Browserstatusleiste zu aktivieren, bei dem per Rechtsklick auch ein Kontextmenü zur Verfügung steht. Das Tool stellt bei Mouseover sowohl den RGB- als auch den Hex-Wert der Farbe unter der aktuellen Cursorposition in der Statusleiste dar.

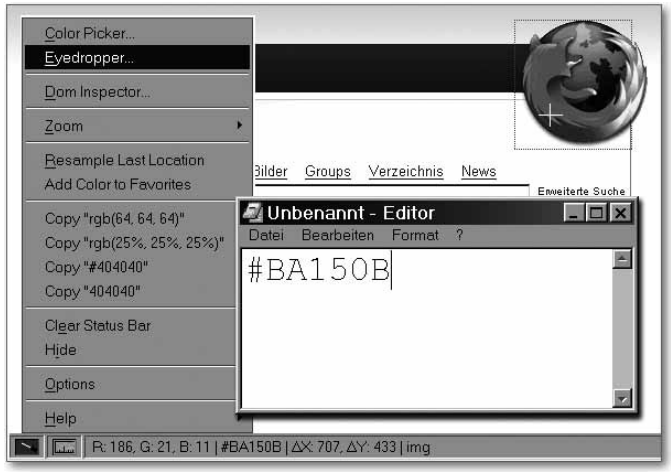

Abbildung 11.10 Der Colorpicker-Eyedropper in Aktion

Interessantes Feature, das die Möglichkeiten anderer Colorpicker übersteigt, ist die Historyliste von ColorZilla, die erfasste Farbwerte in einer Liste zusammenfügt, um so eigene Farbpaletten zusammenzustellen, die als `gpl`-Dateien archiviert und wiederverwendet werden können. Zusätzlich stehen per Dropdown-Menü auch vorgefertigte Paletten zur Verfügung, aus denen Farben gewählt werden können (siehe Abbildung 11.11).

Plattform: Firefox

Status: Freeware-Extension

Beurteilung: Farben per Pipette aufnehmen

Downloadgröße/Festplattenbedarf: 81 KB (xpi-Datei)

Website: www.iosart.com

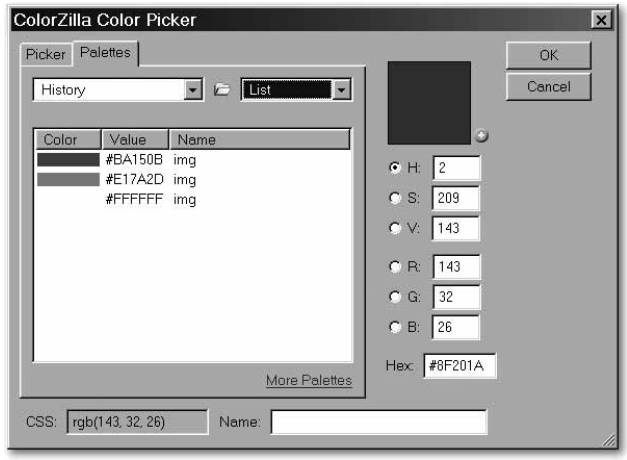

Abbildung 11.11 Die Colorpicker-Palette in der Historyansicht

Colour Spy

Die Freeware **Colour Spy** (Abbildung 11.12) von SilverAge Software bietet eine Pipettenfunktion, die von einem kleinen, frei schwebenden Fenster aus beliebige Farbtöne vom Bildschirm aufnehmen und deren Hex-Wert in die Zwischenablage kopieren kann. Vorteil gegenüber der Firefox Erweiterung ist, dass Color Spy auch außerhalb des Browserfensters arbeiten kann.

Abbildung 11.12 Color Spy bei der Arbeit

Plattform: Windows

Status: Freeware

Beurteilung: Farben per Pipette aufnehmen

Downloadgröße/Festplattenbedarf: 0,3 MB / 0,7 MB

Website: www.silveragesoftware.com

Absolute Color Picker

Der **Absolute Color Picker** (Abbildung 11.13) bietet die Pipettenauswahl von Farben vom Bildschirm (aus jeder laufenden Anwendung) sozusagen »nebenbei« – dies aber mit einer praktischen Lupenfunktion, die pixelgenaues Zielen ermöglicht. Sein Schwerpunkt ist dennoch eher die Unterstützung bei der Zusammenstellung harmonischer Farbkombinationen, wofür eine große Anzahl Paletten bereitgehalten werden, und der Modifikation ausgewählter Farben nach verschiedenen Farbmodellen. So ist es problemlos möglich, von einer Ausgangsfarbe hellere oder dunklere Tönungen oder solche mit höherer oder geringerer Farbsättigung zu erzeugen.

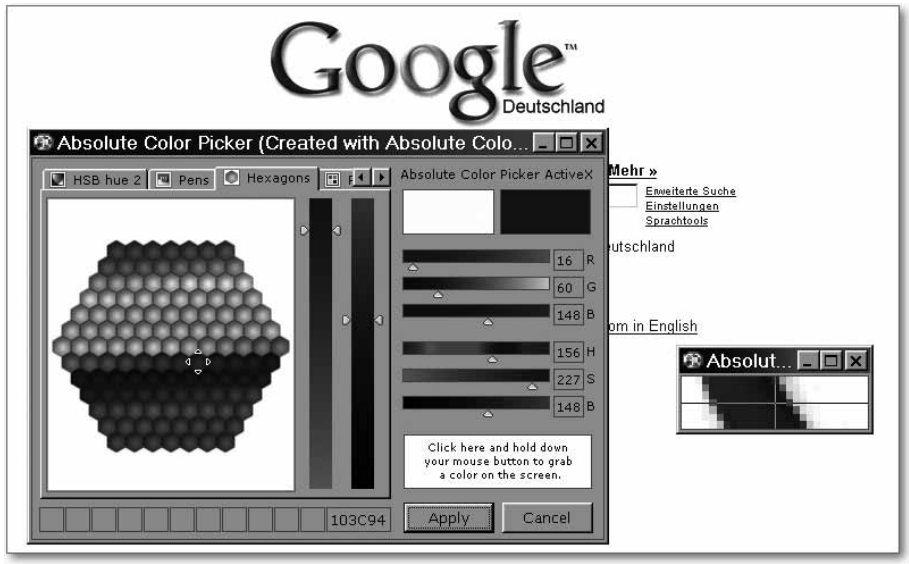

Abbildung 11.13 Absolute Color Picker holt eine Farbe von einer Webseite

Anmerkung: Es wird auch eine kommerzielle, wesentlich leistungsfähigere Version des Color Pickers angeboten (Absolute Color Picker ActiveX Control).

Plattform: Windows

Status: Freeware

Beurteilung: Farben mischen oder per Pipette aufnehmen

Downloadgröße/Festplattenbedarf: 1,28 MB / 1,35 MB

Website: www.eltima.com/

11.3 Tools, Plug-ins und Erweiterungen

DOM-Inspector (Firefox, Mozilla)

Eine mit dem DOM-Explorer der Microsoft Developer Toolbar vergleichbare Lösung gibt es von Chris Neale auch als Erweiterung für Firefox und Mozilla. Mit Hilfe des **DOM-Inspectors** (Abbildung 11.14) kann die Dokumentstruktur einer beliebigen Webseite genau aufgeschlüsselt und einzelne Tag- und Attributeigenschaften ausgelesen werden.

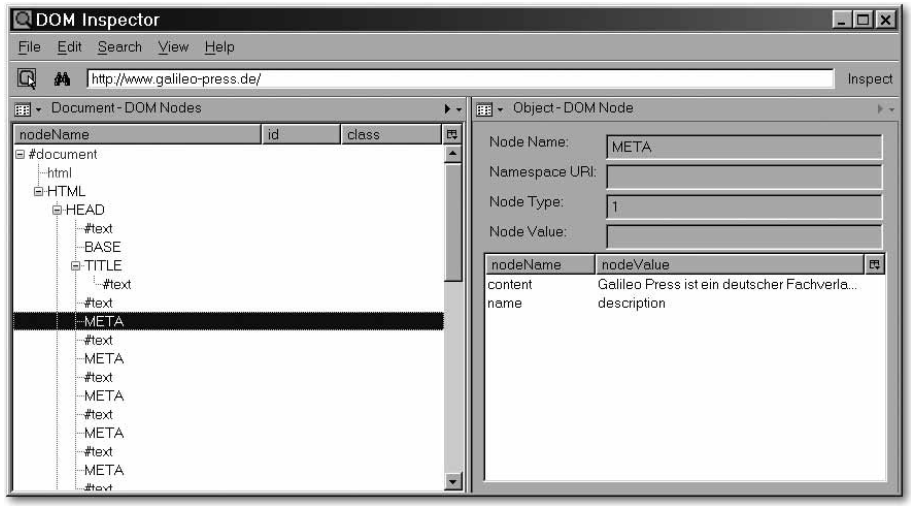

Abbildung 11.14 DOM-Inspector (Firefox, Mozilla)

Eine zusätzliche Hilfe sind die Erweiterungen **InspectThis** von Laurens Mackay (firefox.mackay.dyndns.info) oder das vergleichbare **Inspect Element** von Nicolas Moreau (stricto-sensu.net), die den DOM-Inspector gezielt bei Rechtsklick auf ein Element per Kontextmenü aufrufen (je ca. 11 KB, xpi-Files).

Plattform: Firefox, Mozilla

Status: Freeware-Extension

Beurteilung: Gezielte Information über die Dokumentstruktur

Downloadgröße/Festplattenbedarf: 272 KB (xpi-File)

Website: www.mozilla.org/projects/inspector/

Firebug

Firebug von Joe Hewitt bietet eine komfortable Debugging-Umgebung für Firefox, die, anders als die bordeigene JavaScript-Konsole zwischen JavaScript- und CSS-Fehlern unterscheiden und obendrein ähnlich wie bei einem DOM-Inspektor, jedoch intuitiver, per Klick Informationen über beliebige Objekte der Webseite preisgeben kann (siehe Abbildung 11.15).

Abbildung 11.15 Firebug bein Inspizieren eines Elements (Firefox, Mozilla)

Plattform: Firefox, Mozilla

Status: Freeware-Extension

Beurteilung: Gute Hilfe bei Fehlersuche

Downloadgröße/Festplattenbedarf: 33 KB (xpi-File)

Website: www.joehewitt.com

11.4 Textbrowser und Textbrowser-Simulationen

Wer einen Eindruck erhalten möchte, wie die entworfene Website aus Sicht eines Textbrowsers aussieht, kann, wenn er nicht den Original-Lynx-Browser installieren möchte, auf eine der entsprechenden Simulationen zurückgreifen.

Die Textansicht einer Webseite ist ungeheuer ernüchternd – auf die Wiedergabe von Farben, Bildern und CSS-Textformatierung muss hier komplett verzichtet werden. Auch repositionierte oder versteckte (Stichwort: `display:none`) Inhalte erscheinen hier »as is« in Quelltextreihenfolge.

Es ist jedoch nützlich zu wissen, was bestimmte Art von User Agents von der Seite wahrnehmen und wie sich die Informationsstruktur aus diesem Blickwinkel bewährt. Es gibt drei Arten von User Agents, die mit einer Quasi-Textansicht arbeiten (müssen):

- Screenreader
- Braille-Reader
- Suchmaschinen

Es ist nicht nur lehrreich, sondern eigentlich unverzichtbar, gelegentlich diese Perspektive einzunehmen, um Defizite eines Designs aufzudecken – sowohl in der Accessibility (speziell für Sehbehinderte) als auch gleichzeitig in der Zugänglichkeit für Suchmaschinen.

Lynx Browser – das Original

Der **Lynx** (Abbildung 11.16) ist auch heutzutage noch für verschiedene Plattformen ohne Weiteres erhältlich. Es gibt Versionen für Linux, Unix, VMS, DOS, Windows (95/98/NT) – inoffiziell auch unter Windows 2000/XP.

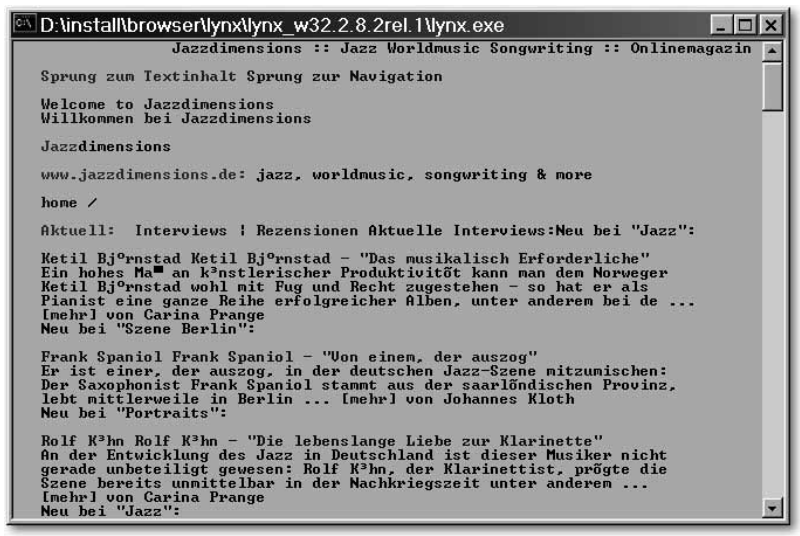

Abbildung 11.16 Lynx unter Windows – reichlich karg

Plattform: Windows, DOS, Linux, Mac OS

Status: Open-Source-Software

Beurteilung: Das Original unter den Textbrowsern

Downloadgröße/Festplattenbedarf: ca. 0,7 MB/1,5 MB (je nach Plattform)

Website: http://lynx.isc.org und verschiedene Quellen
(siehe Anhang Ressourcen)

Yellowpipe Lynx Viewer

Als Firefox- bzw. Internet Explorer-Erweiterung realisiert ist der **Lynx-Viewer** von Yellowpipe (Abbildung 11.17), der eine Textversion der aktuell im Browserfenster gezeigten Webseite in einem separat geöffneten Fenster generiert. Die Anzeige ist allerdings reine Optik – im Gegensatz zu Lynx können hier keine Links verfolgt werden, die Anzeige ist statisch. Die Erweiterung wird über das Kontextmenü aktiviert.

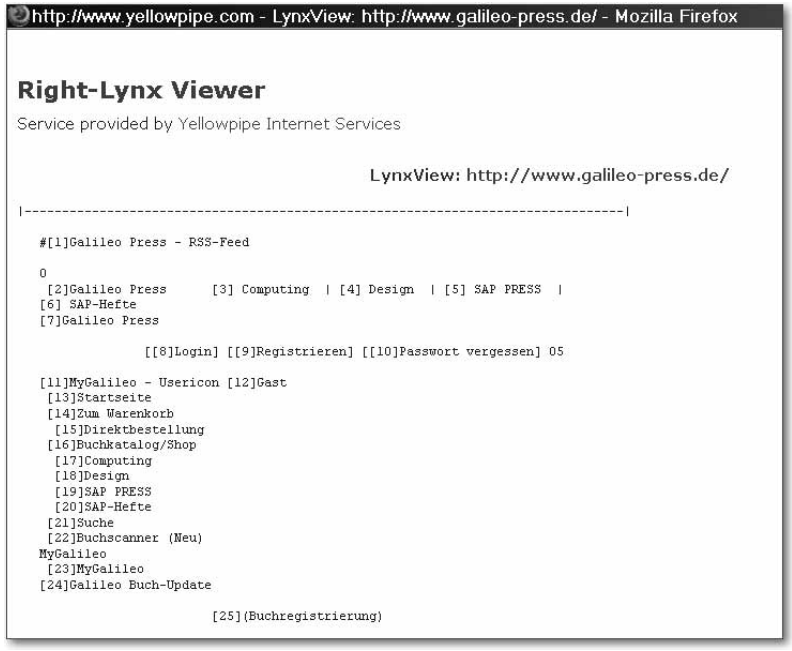

Abbildung 11.17 Lynx-Simulation von Yellowpipe (Firefox-Erweiterung)

Yellowpipe bietet auch andere Firefox-Extensions und weitere kostenlose Tools an (Window Resizer, Color Picker, Color Converter, META Tag-Generator, robots.txt-Generator u.v.a.).

Plattform: Firefox, Internet Explorer 6.x

Status: Freeware-Extension

Beurteilung: Brauchbare Simulation

Downloadgröße/Festplattenbedarf: 25,9 KB (xpi-File)

Website: www.yellowpipe.com

Delorie Lynx Viewer

Direkt über die Firmen-Website **www.delorie.com** oder über die **Web Accessibility Toolbar** (**Extras > Simulationen > Lynx Viewer**) ist die sehr gute Lynx-Simulation von Delorie zu erreichen (siehe Abbildung 11.18).

Einziger Nachteil der Simulation besteht darin, dass auf der zu betrachtenden Website im Wurzelverzeichnis, um Missbrauch vorzubeugen, eine Datei **delorie.htm** abgelegt sein muss (diese braucht keine weiteren Inhalte zu besitzen). Wird diese Datei nicht gefunden, erhält man die Nachricht »Not Permitted«.

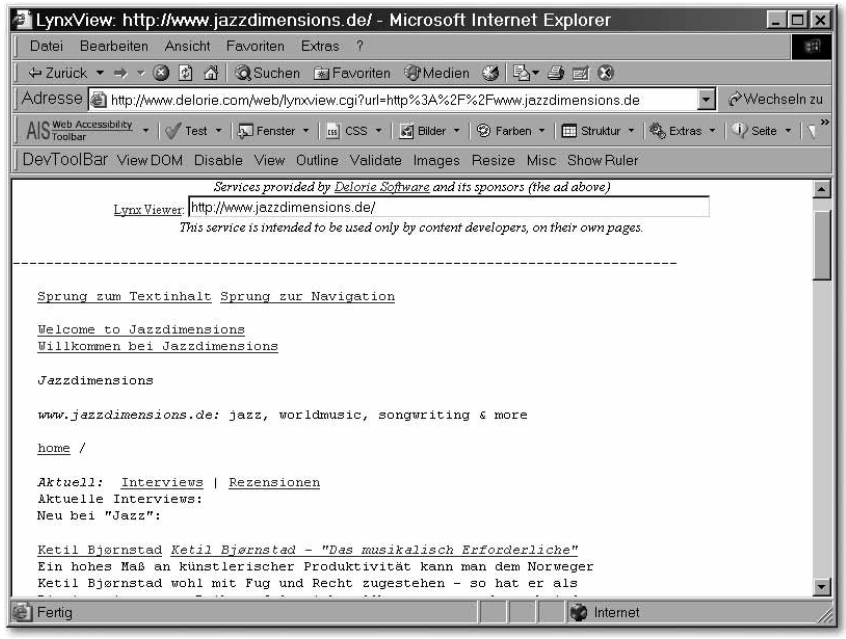

Abbildung 11.18 Lynx Viewer bei www.delorie.com

Plattform: alle Browser

Status: Online-Service

Beurteilung: Gute Simulation mit verfolgbaren Links

Downloadgröße/Festplattenbedarf: entfällt

Website: www.delorie.com

Dieses Kapitel bietet einen Abriss der historischen Entwicklung des HTML-Vokabulars und der verschiedenen Sprachfamilien und Versionen von HTML Level 0 bis XHTML 2.0. Vorgestellt werden darüber hinaus auch die Versionen von HTML für mobile Endgeräte.

12 HTML-Sprachfamilien

Im Laufe seiner Entwicklung hat HTML zahlreiche Wandlungen und Erweiterungen erfahren, die sich in den Vokabularen der verschiedenen Sprachversionen niederschlägt.

12.1 Die Vergangenheit: HTML-Versionen 1.0 bis 3.2

Gerade in seiner Frühzeit zeigt sich HTML als eine sich dynamisch entwickelnde Sprache. Einige der damals aus Machbarkeitsgründen verworfenen Konzepte sind dabei nicht nur von historischem Interesse.

12.1.1 HTML Level 0

HTML entstand im Ursprung lediglich »in Anlehnung an« SGML, erst später (zu HTML 2.0) wurde tatsächlich eine DTD gemäß SGML definiert. Der als »Level 0« bezeichnete Sprachumfang spiegelt den Stand des Konzepts von Tim Berners-Lee aus dem Jahr 1989/1990 wider, der sich noch an textorientierte Browser richtete: Einzig der `<TITLE>`-Container war für ein HTML-Dokument zwingend vorgeschrieben, `<HTML>`-, `<HEAD>` und `<BODY>`-Elemente existierten noch nicht.

Anfänglich gab es nur die von Berners-Lee (auf NeXT) geschriebene Browsersoftware, die er »World Wide Web«-Browser nannte (Abbildung 12.1); erst ab Ende 1990 erfolgten Versuche mit weiteren Browsern für verschiedene Plattformen.

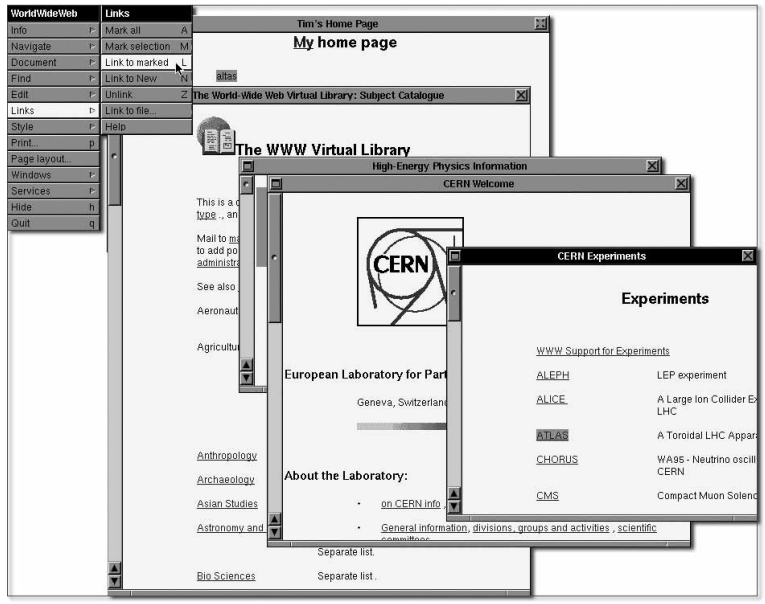

Abbildung 12.1 Tim Berners-Lees »World Wide Web« (ca. 1993)

HTML »Level 0« umfasste 23 Elemente:

`<A> <ADDRESS>`
`<DD> <DIR> <DL> <DT>`
`<H1> <H2> <H3> <H4> <H5> <H6>`
`<ISINDEX>`
`<LI> <LISTING>`
`<MENU>`
`<NEXTID>`
`<OL>`
`<P> <PLAINTEXT>`
`<TITLE>`
`<UL> <XMP>`

Tabelle 12.1 Elemente von HTML Level 0

Anmerkungen: Vorgesehen, aber damals noch nicht verwirklicht, war eine Funktionalität, die dem späteren `<BASE>`-Tag entsprach. Die angedachte Option, Textabschnitte (*highlighted phrase*) mit Tags in Form `<HP1>`, `<HP2>`, ... hervorzuheben, wurde fallengelassen. Der `<P>`-Tag wurde von Berners-Lee noch im Sinne eines Separators zwischen Absätzen verstanden.

12.1.2 HTML 1.0

HTML 1.0, auch »HTML Level 1« genannt (die Bezeichnung HTML 1.0 wurde erst rückblickend verliehen), entstand ab 1992 aus der Zusammenarbeit von Berners-Lee mit Dan Connolly[1], um einige erkannte Defizite von »Level 0« zu beheben. Die Idee eines <HTML>-Containers und Elemente für <HEAD> und <BODY> kamen hinzu sowie der <IMG>-Tag und einige logische Elemente mit Präsentationswirkung.

Der Sprachumfang umfasste 47 Elemente (die gegenüber »Level 0« hinzugekommenen Elemente sind in der Liste fett markiert):

<A> <ADDRESS>
**<B> <BASE> <BLOCKQUOTE> <BODY> **
<CITE> <CODE>
<DD> <DFN> <DIR> <DL> <DT>
<EM>
<H1> <H2> <H3> <H4> <H5> <H6> **<HEAD> <HTML> <HR>**
<I> <IMG> <ISINDEX>
<KBD> <KEY> <LI> **<LINK>** <LISTING>
<MENU>
<NEXTID>
<OL>
<P> <PLAINTEXT> <PRE>
<S> <SAMP> <STRONG>
<TITLE> **<TT>**
<U> <UL> **<VAR>** <XMP>

Tabelle 12.2 Elemente von HTML 1.0

Anmerkungen: Die Entwicklung von HTML war in jener Zeit fließend. Man unterschied weniger feste Sprachversionen als so genannte »Levels«, die sich an den erforderlichen Fähigkeiten der Browser orientierten. Ein »Level 0«-Browser war nach dieser Lesart ein rein textbasierter Browser, ein »Level 1«-Browser sollte zusätzlich den Umgang mit Bildern und weiteren Elementen beherrschen. In Planung waren ein »Level 2« (das einfache Formulare umfassen sollte) sowie ein »Level 3« (gemeint war die später verworfene Studie HTML+).

12.1.3 HTML 2.0

Aus den Arbeiten von Berners-Lee und Connolly zu HTML »Level 2« und unter Berücksichtigung einiger (weniger) Vorschläge von Dave Raggett[2] zu HTML+ ent-

1 Mehr über Dan Connolly: http://www.w3.org/People/Connolly/.

2 Mehr über Dave Raggett siehe: www.w3.org/People/Raggett/.

stand die Spezifikation **HTML 2.0**, als Draft vorgestellt auf der ersten Web Conference in Genf im Jahr 1994. Die endgültige Spezifikation wurde im November 1995 verabschiedet und als RFC 1866 der IETF (Internet Engineering Task Force) veröffentlicht.

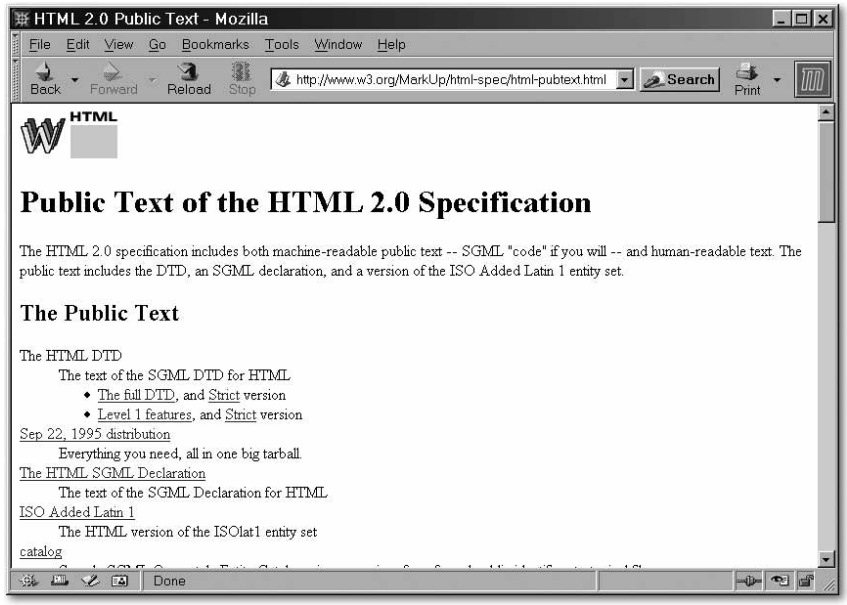

Abbildung 12.2 Die erste wirkliche Spezifikation zu HTML

HTML 2.0 lag erstmals eine DTD zugrunde. Die wichtigsten Neuerungen bestanden in Formularen und der Möglichkeit, einem Dokument Metainformationen hinzuzufügen. Die Tagkollektion von HTML wuchs in HTML 2.0 nur geringfügig auf 52 Elemente (Neuzugänge gegenüber HTML 1.0 fett markiert):

<A> <ADDRESS>
<B> <BASE> <BLOCKQUOTE> <BODY>
<CITE> <CODE>
<DD> <DFN> <DIR> <DL> <DT>
<EM> **<FORM>**
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> <IMG> **<INPUT>** <ISINDEX>
<KBD> <LI> <LINK> [<LISTING>]
<MENU> **<META>**
<NEXTID>

Tabelle 12.3 Elemente von HTML 2.0

<OL> **<OPTION>**
<P> [<PLAINTEXT>] <PRE>
<S> <SAMP> **<SELECT>** <STRONG>
<TEXTAREA> <TITLE> <TT>
<U> <UL> <VAR> [<XMP>]

Tabelle 12.3 Elemente von HTML 2.0 (Forts.)

Anmerkungen: Erstmals wurde das Prinzip der »Missbilligung« existierender HTML-Elemente angewandt – so wurden die von Berners-Lee in »Level 0« eingeführten Tags <LISTING>, <PLAINTEXT> und <XMP> als »deprecated« bezeichnet (in der Tabelle durch eckige Klammern markiert). Der bislang als »Trennungsmarkierung« (vergleichbar mit <BR> und <HR>) verstandene <P>-Tag wurde nun offiziell als Container definiert, wenn auch mit optionalem Endtag.

12.1.4 HTML 3.2

Als nächster verabschiedeter Standard folgte am 14. Januar 1997 auf HTML 2.0 die Version **HTML 3.2**. Diese Version enthielt eine Vielzahl von Tags, die von der mächtiger werdenden Browserindustrie (in jener Zeit im Wesentlichen durch Netscape) propagiert und von damaligen Webdesignern bereits aufgegriffen worden waren und die meist Präsentation betrafen (Farben, Textausrichtung, Schriftart und -größe). Ebenfalls neu war die Möglichkeit, Tabellen, Applets, Imagemaps, Scripte (für Netscapes JavaScript) und Stylesheets (für das damals neue CSS) einzusetzen. HTML 3.2 orientierte sich also an geschaffenen Fakten – die ursprünglich vom W3C geplante Version 3.0 musste verworfen werden.

Die Zahl der Elemente wächst in HTML 3.2 auf 70 an (Neuzugänge fett):

<A> <ADDRESS> **<APPLET>** **<AREA>**
<B> <BASE> **<BASEFONT>** <BIG> <BLOCKQUOTE> <BODY>
<CAPTION> **<CENTER>** <CITE> <CODE>
<DD> <DFN> <DIR> **<DIV>** <DL> <DT>
<EM> **<FONT>** <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> <IMG> <INPUT> <ISINDEX>
<KBD> <LI> <LINK> [<LISTING>]
<MAP> <MENU> <META>
<OL> <OPTION>
<P> **<PARAM>** [<PLAINTEXT>] <PRE>

Tabelle 12.4 Elemente von HTML 3.2

<S> <SAMP> **<SCRIPT>** <SELECT> **<SMALL>** **<STRIKE>** <STRONG> **<STYLE>** **<SUB>** **<SUP>**
<TABLE> **<TD>** <TEXTAREA> **<TH>** <TITLE> **<TR>** <TT>
<U> <UL> <VAR> [<XMP>]

Tabelle 12.4 Elemente von HTML 3.2 (Forts.)

Anmerkungen: Von HTML 2.0 zu HTML 3.2 geschehen in Bezug auf Elemente und deren Attribute einige kleinere Bereinigungen am Vokabular – so entfällt das <NEXTID>-Element und das urn- bzw. methods-Attribut von <A>. Die hier wie in der Vorgängerversion HTML 2.0 als »missbilligt« gekennzeichneten Elemente <LISTING>, <XMP> und <PLAINTEXT> entfallen erst in der folgenden Version HTML 4.0. <S> sollte durch <STRIKE> ersetzt werden. Von der Übernahme *aller* damaligen Netscape-Vorschläge sah man (in der Regel zum Glück) ab – so wurde der berüchtigte <BLINK>-Tag nie zum Standard; hiermit war jedoch auch das Prinzip des nicht-konformen, »proprietären« Tags geboren.

12.2 Verpasste Chancen: HTML+ und HTML 3.0

Zweimal in der Frühgeschichte von HTML wurde versucht, die Entwicklung der Sprache abseits von Sachzwängen gezielt zu steuern – das erste Mal im November 1993 in Form eines durch Dave Raggett entwickelten, als **HTML+** bezeichneten Vorschlags, dessen Name die Intention widerspiegelt, die in der Erweiterung der Funktionalität der damals vorherrschenden Version HTML 1.0 bestand. (Damals gab es die Bezeichnung »HTML 1.0« eigentlich nicht – man bezeichnete das HTML jener Zeit erst nach Aufkommen von HTML 2.0 in dieser Weise.)

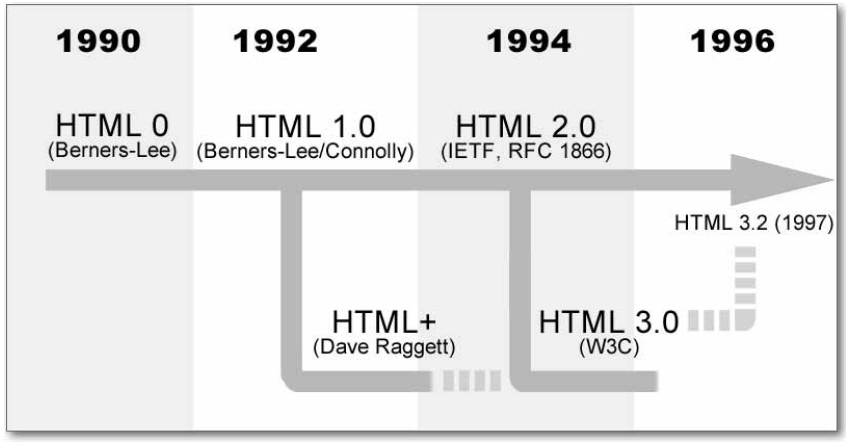

Abbildung 12.3 HTML und seine Seitenlinien HTML+ und HTML 3.0

Eine verfrühte Vision: HTML+

Der Entwurf zu HTML+ gab sich mit der Einführung von Formularen, Tabellen und der Möglichkeit, mathematische Formeln darzustellen, revolutionär und fortschrittlich (dies vielleicht zu sehr). Der damals beginnende Boom der Browsersoftware schuf jedoch Fakten abseits des geplanten Vorhabens – hierzu zählten Cello, Chimera, Erwise, Midas, Viola, der NeXT-Browser und viele weitere, inzwischen vergessene Produkte, vor allem aber der für eine ganze Reihe von Plattformen vorliegende »Mosaic« als erster nicht-textorientierter Browser.

Browserarchiv

Sehr viele der hier angesprochenen historischen Browser und alle Versionen der aktuellen Browser sind im Archiv von **www.evolt.org** zu finden. Zu Viola finden Sie Informationen unter **www.xcf.berkeley.edu/~wei/viola/**.

Noch während der laufenden Planung von HTML+ erfolgte im April 1994 durch die damalige HTML-Arbeitsgruppe[3] der IETF (Internet Engineering Task Force, **www.ietf.org**) eine Einigung auf den Standard **HTML 2.0**, der als RFC 1866 veröffentlicht wurde. Hierbei wurden zwar auch einige Ideen aus HTML+ aufgegriffen, gleichzeitig jedoch begann das Einwandern von Elementen, die den Marketingüberlegungen der Browserhersteller entstammten – ein Umstand, der später während des »Browserkrieges« zwischen Microsoft und Netscape in einer stark eingetrübten und inkohärenten Tagkollektion resultierte (die daher polemisch auch als »*tag soup*« bezeichnet wird).

HTML 3.0

Der zweite Versuch eines Befreiungsschlags erfolgte März 1995, inzwischen unter der Ägide des Dezember 1994 gegründeten **W3C** (World Wide Web Consortium) mit dem Entwurf zu **HTML 3.0**, das als Standard HTML 2.0 ablösen sollte. Verschiedene Konzepte und Elemente aus HTML+ wurden hierfür wieder aufgegriffen, der sich abzeichnenden Entwicklung des Aufkommens von Präsentationselementen sollte durch das Propagieren von Stylesheets als Alternative entgegengesteuert werden.

Doch erneut wurde das ehrgeizige Projekt durch die von den Browserherstellern geschaffenen Fakten überrollt: Als Kompromiss zwischen Herstellern und W3C wurde der Standard **HTML 3.2** verabschiedet, der trotz der höheren Versionsnummer gegenüber HTML 3.0 eigentlich einen Rückschritt bedeutete. Mit ihm wurde als Fakt festgeschrieben, was später mühsam wieder eliminiert werden

3 Diese Arbeitsgruppe wurde 1996 aufgelöst, da ihre Aufgaben durch das W3C übernommen worden waren.

musste: die Vermischung von Struktur und Präsentation in HTML. Immerhin gelang es in Folge, das W3C als Gegengewicht gegenüber der Willkür der Browserhersteller zu etablieren.

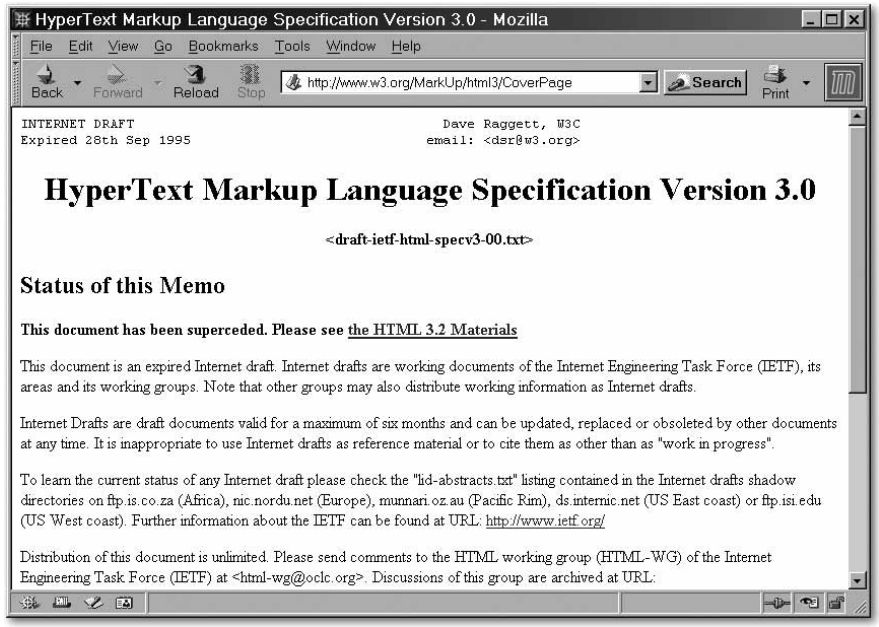

Abbildung 12.4 Der Spezifikationsvorschlag zu HTML 3.0

Ein kurzer Blick auf die Tagkollektionen beider Entwürfe lohnt durchaus. Waren auch einige Ideen ausufernd und redundant, so tauchen doch bemerkenswert viele Vorschläge in späteren Versionen von HTML und sogar XHTML wieder auf – ein Beispiel hierfür ist der `<L>`-Tag (Zeilencontainer) aus HTML+, der als `<l>`-Container in XHTML 2.0 seine Wiederauferstehung feiern darf, oder (ebenfalls aus HTML+) der neben dem `<IMG>`-Tag propagierte `<IMAGE>`-Container, der im Ansatz mit dem späteren `<object>`-Container verwandt ist.

12.2.1 HTML+ (HTMLPlus)

Die Tagliste des Diskussionsvorschlags zu **HTML+** vom November 1993 umfasst 78 Elemente – wesentlich mehr als in HTML 2.0 enthalten waren und noch immer mehr, als später in HTML 3.2 verwirklicht werden sollten:

`<A> <ABBREV> <ABSTRACT> <ACRONYM> <ADDED> <ADDRESS> <ARG>`
`<B> <BASE> <BLOCKQUOTE> <BODY> <BOX>   <BYLINE>`

Tabelle 12.5 Elemente von HTML+

<CAPTION> <CHANGED> <CITE> <CMD> <CODE>
<DD> <DFN> <DIR> <DL> <DT>
<EM> <FIG> <FOOTNOTE> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HTMLPLUS>
<I> <IMAGE> <IMG> <INPUT> <ISINDEX>
<KBD> <L> <LI> <LINK> <LIT>
<MARGIN> <MATH> <MENU>
<NEXTID> <NOTE>
<OL> <OPTION> <OVER>
<P> <PERSON> <PRE>
<Q> <QUOTE> <RENDER> <REMOVED>
<S> <SAMP> <SELECT> <STRONG> <SUB> <SUP>
<TAB> <TABLE> <TD> <TEXTAREA> <TH> <TITLE> <TR> <TT>
<U> <UL> <VAR>

Tabelle 12.5 Elemente von HTML+ (Forts.)

Eine Reihe Elemente wie <MATH>, die es nicht nach HTML 2.0 »schafften«, tauchen (wenn auch mit anderen Kindelementen) im Entwurf zu HTML 3.0 wieder auf. Ebenso wird die Idee hinter <CHANGED>, <REMOVED> und <ADDED>, die zur Markierung von Editierungsvorgängen dienen sollten, bei HTML 3.0 als <DEL> und <INS> ansatzweise wieder aufgegriffen, aber erst in HTML 4.0 realisiert (und bis heute nicht allgemein unterstützt). Der <RENDER>-Tag war ein interessanter Ansatz, ohne Styledefinitionen mit unbekannten (ansonsten ignorierten) Elementen umzugehen, der allerdings wieder in der Schublade verschwand.

Interessant ist auch, welche Elemente *nicht* in dieser Liste auftauchen – so ist in diesem Entwurf der berüchtigte, aber offiziell noch bis HTML 3.2 mitgeschleppte <PLAINTEXT>-Tag gar nicht erst vorgesehen. Ob hingegen der Vorschlag, das Wurzelelement <HTML> in <HTMLPLUS> umzutaufen, Aussicht auf Erfolg gehabt hätte, ist spekulativ. Ein Wurzelelement war damals in HTML-Dokumenten ohnehin nicht zwingend vorgeschrieben.

Benennung des Wurzelelements

Heutzutage mutet die Absicht einer solchen Umtaufe wie ein Sakrileg an – auch in XHTML hält man an <html> als Wurzelelement fest. Da aber gewöhnlich das Wurzelelement einer Sprache entsprechend der Dokumenttyp-Definition benannt wird (die HTMLPLUS heißt), war der Vorstoß im Grunde folgerichtig.

Die, hier erstmals vorgeschlagene, Implementierung von Tabellen war, trotz gleichlautender Tagbezeichner *nicht* identisch mit der in HTML 2.0 umgesetzten;

so markierte der <TR>-Tag als *row separator,* vergleichbar mit dem
-Tag, das Ende einer Tabellenzeile und war als leeres Element definiert. In diesem Fall muss die verwirklichte Version als fortschrittlicher gelten.

Was in der reinen Tagliste nicht sichtbar ist, sind einige interessante, auf Attributebene geplante Ideen wie das PLAIN-Attribut von <UL>, das Listen ohne Aufzählungssymbole (Bullets) ermöglichen sollte, das EFFECT-Attribut von <A>, das das Linkverhalten steuern sollte, das SEETHRU-Attribut von <IMG>, um Transparenz-Farben zu definieren[4], oder einige Attribute und -werte bei Formularen, für die vergleichbare Funktionalität erst wieder bei XForms auftaucht. Auch »globale« Attribute gibt es – jedes Element sollte ein ID- und (die Mehrzahl der Elemente) ein INDEX-Attribut tragen können; Letzteres geplant als Hilfsmittel für eine angedachte automatische Indexerstellung mit Primär- und Sekundärschlüsselworten.

12.2.2 HTML 3.0

Die von Dave Raggett für HTML+ geleistete Konzeptionsarbeit floss zu großen Teilen in den Entwurf zu **HTML 3.0** ein, der als Ablösung für die Spezifikation **HTML 2.0** propagiert wurde. Die wohl wichtigste hier vorgeschlagene Neuerung ist der <STYLE>-Container zur Beherbergung von Styleanweisungen. Damals war noch an »DSSSL lite« als Stylesprache gedacht, später als »DSSSL Online« bezeichnet (CSS Level 1wurde erst Dezember 1996 verabschiedet).

Der Entwurf umfasst insgesamt 104 Elemente – mehr als alle anderen[5] HTML-Versionen:

<A> <ABBREV> <ABOVE> <ACRONYM> <ADDRESS> <ARRAY> <ATOP> <AU>
<B> <BANNER> <BAR> <BASE> <BELOW> <BIG> <BLOCKQUOTE> <BODY> <BODYTEXT> <BOX> <BQ>   <BT>
<CAPTION> <CHOOSE> <CITE> <CODE> <CREDIT>
<DD> <DDOT> <DEL> <DFN> <DIR> <DIV> <DL> <DOT> <DT>
<EM> <FIG> <FIGTEXT> <FN> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HAT> <HEAD> <HR> <HTML>
<I> <IMG> <INPUT> <INS> <ISINDEX> <ITEM>
<KBD> <LANG> <LEFT> <LH> <LI> <LINK>
<MATH> <MENU> <META>
<NEXTID> <NOTE>

Tabelle 12.6 Elemente von HTML 3.0

4 Damals war GIF das dominierende Grafikformat im Web, GIF-Transparenz nach Gif89a jedoch noch nicht allgemein unterstützt. Daher vielleicht dieser eigentlich überflüssige Ansatz.

5 Die umfangreichste, tatsächlich eingeführte HTML-Version ist »HTML 4.0 frameset« mit 91 Elementen.

<OF> <OL> <OPTION> <OVER> <OVERLAY>
<P> <PERSON> <PRE>
<Q> <RANGE> <RIGHT> <ROOT> <ROW>
<S> <SAMP> <SELECT> <SMALL> <SPOT> <SQRT> <STRONG> <STYLE> <SUB> <SUP>
<T> <TAB> <TABLE> <TD> <TEXTAREA> <TH> <TILDE> <TITLE> <TR> <TT>
<U> <UL> <VAR> <VEC>

Tabelle 12.6 Elemente von HTML 3.0 (Forts.)

Der überraschend große Sprachumfang rührt von jenen Elementen her, die zur Darstellung mathematischer Formeln vorgesehen waren – diese Gruppe allein umfasst in HTML 3.0 rund 20 Elemente. Hierfür wurde der Container <MATH> aus HTML+ hinübergerettet, sein Inhaltsmodell jedoch vollständig umgearbeitet und erweitert (<ARRAY> wurde übernommen und erweitert; <OVER> aus HTML+ entspricht <ABOVE> bei HTML 3.0 – hier erschöpfen sich bereits die Ähnlichkeiten).

Die Einbeziehung von Formeldarstellung in HTML wurde bereits in HTML 3.2 fallengelassen und später separat in Form von **MathML**[6], einer XML-Anwendung, wieder aufgegriffen. Durch Integration[7] von MathML mit XHTML 1.1 lässt sich die für HTML 3.0 geplante Funktionalität sogar noch weit übertreffen.

Anmerkungen: Vier Elemente waren im Vorschlag zu HTML 3.0 nur aus Kompatibilitätsgründen zu HTML 2.0 enthalten (<BLOCKQUOTE>, <DIR>, <MENU> und <NEXTID>). Es existieren daher zu HTML 3.0 zwei DTD-Vorschläge – ein kompatibler und ein orthodoxer, *HTML 3.0 strict* genannter, der auf die vier vorgenannten Elemente verzichtet.

Sehr weit ging das Vertrauen in die Abwärtskompatibilität allerdings nicht – es wurde über einen eigenen MIME-Contenttyp und eine spezielle Dateiendung `.html3` oder `.ht3` nachgedacht, um »alte« User Agents von HTML 3.0-Dokumenten fernhalten zu können. Beginnend mit HTML 3.2 verwendete man eine DOCTYPE-Deklaration zur besseren Unterscheidbarkeit.

12.3 HTML 4.0 und HTML 4.01

HTML 4.01 (von April 1998) stellt die derzeit aktuelle offiziell vom W3C abgesegnete Version von HTML dar. Streng genommen handelt es bei dem minimalen Versionssprung von **HTML 4.0** lediglich um einige Fehlerbereinigungen gegen-

6 MathML, Mathematical Markup Language (siehe: www.w3.org/Math/). Inzwischen liegt diese (seit 2002) auch in der Version MathML 2.0 vor (siehe: www.w3.org/TR/MathML2/).

7 Hierfür existiert auch eine eigene DTD, die zusätzlich noch SVG beinhaltet: »XHTML 1.1 plus MathML 2.0 plus SVG 1.1« (siehe: www.w3.org/TR/XHTMLplusMathMLplusSVG/).

über der im Dezember 1997 verabschiedeten Urversion HTML 4.0. Für die nachfolgende Referenz wird Version 4.01 als maßgeblich betrachtet.

Welche Unterschiede gibt es zwischen HTML 4.0 und HTML 4.01?

In der HTML 4.0 Spezifikation war das `name`-Attribut für die Elemente `<FORM>` und `<IMG>` nicht definiert. HTML 4.01 behebt dieses Manko.

Voraussichtlich wird die Version 4.01 die letzte, offiziell vom W3C entwickelte, auf SGML basierende Sprachversion von HTML bleiben – das Konsortium hat sich auf das von XML abgeleitete XHTML als ablösenden Standard geeinigt.

Weiterentwicklung von HTML abseits vom W3C?

Es gibt Bestrebungen, HTML weiterzuentwickeln, populär als »HTML 5.0« bezeichnet. In die als »Web Applications 1.0« bezeichnetete Spezifikation sollen auch Funktionalitäten von XHTML einfließen. Die Arbeiten werden u.a. von Apple und verschiedenen Browserherstellern, darunter Opera und Mozilla, vorangetrieben (siehe: **www.whatwg.org**).

Die Dokumenttyp-Varianten

Etwa mit dem geplanten Erscheinen von HTML 3.0 (1995) kann man den Zeitpunkt festmachen, seitdem das W3C verstärkt versucht, eine konsequente Trennung von Struktur und Präsentation für HTML auch in den Spezifikationen durchzusetzen und Stylesheets anstelle von Präsentationstags und -attributen zu propagieren. Da sich eine vollständige Entfernung der entsprechenden, im Zuge der Bereinigung als »missbilligt« (deprecated) bezeichneten Sprachelemente aus der Spezifikation zu HTML 4.0 wegen deren weiter Verbreitung jedoch nicht ad hoc durchführen ließ, entschied man sich stattdessen dafür, die zugrunde liegenden Dokumenttypdefinitionen in drei Varianten[8] zu erstellen:

- **HTML 4.0 strict**
  Diese Variante entspricht den orthodoxen Vorstellungen des W3C. Ein gemäß dieser DTD valides Dokument verzichtet auf Frames und dokumentinterne Präsentationsangaben. *HTML 4.0 strict* verfügt daher über einen verglichen mit den beiden anderen Varianten verringerten Sprachumfang.
- **HTML 4.0 transitional**
  Diese, offiziell als »Übergangsversion« bezeichnete Variante enthält zusätzlich zum *strict*-Vokabular die der verbreiteten Sprachpraxis entsprechenden missbilligten Elemente und Attribute, wie erstere jedoch nicht die Elemente zur Erstellung von Framesets.

8 Die Idee einer DTD-Variante »strict« bestand bereits zu Zeiten des Drafts zu HTML 3.0, wurde aber damals nicht formal vollzogen.

- **HTML 4.0 frameset**
  HTML 4.0 frameset enthält alle Sprachelemente der eben beschriebenen *transitional*-Variante und erweitert diese lediglich um die Frameset-Elemente. Sie ist daher ebenfalls als *transitional* zu betrachten. Ihre Verwendung ist auf Frameset-Dokumente begrenzt – der Umfang des Vokabulars ist lediglich der freien Gestaltbarkeit eines möglichen `<NOFRAMES>`-Containerinhalts geschuldet.

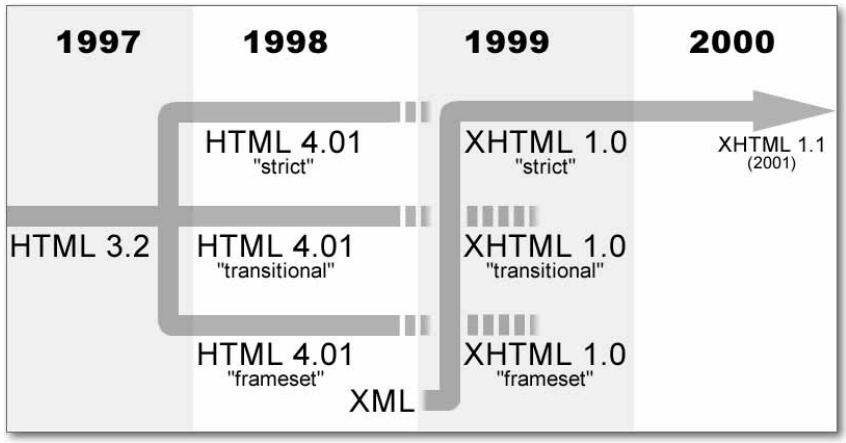

Abbildung 12.5 Die Aufspaltung der HTML-Spezifikation in drei Zweige

Dokumenttypen von HTML 4.01, XHTML 1.0 und XHTML 1.1

Auch in der Revision HTML 4.01 besteht die Parallelität mehrerer Varianten der DTD fort und spiegelt sich ebenfalls in **XHTML 1.0** wider, das ja eine »vollständige Nachbildung« von HTML mittels XML darstellt. Aus Gründen der Abwärtskompatibilität wurden die unterschiedlichen Vokabulare dort daher beibehalten.

12.3.1 HTML 4.01 strict

Die Tagliste von *HTML 4.01 strict* enthält all jene Tags und Attribute des HTML-Vokabulars, die vom W3C gebilligt werden. Dies ist im Sinne der rigorosen Auslegung der Verwendung von HTML als reine Strukturspache zu verstehen, wobei alle Darstellungsanweisungen in Form von CSS erfolgen sollen.

Strenge Trennung zwischen Struktur und Darstellung

Dies wird seit dem Vorschlag zu HTML 3.0 (März 1995) propagiert, spiegelt sich hier aber erstmalig in einem Tag-Vokabular wider. HTML 3.0 bezog sich allerdings noch auf eine (nie realisierte) Variante von DSSSL als Stylesheetsprache. CSS existiert erst seit 1996.

Konsequenterweise entfallen daher im *strict*-Vokabular jene Sprachelemente, die reinen Präsentationszwecken dienen – dies sind neben diversen Attributen am prominentesten, <FONT> und <CENTER> –, aber auch die Frameset-Tags wie <FRAMESET>, <FRAME> und <IFRAME>. Objekteinbindung in Seiten soll konsequent über den <object>-Tag erfolgen, weswegen auch <APPLET> im *strict*-Vokabular fehlt.

Dass Präsentationstags wie <b> und <i> dennoch enthalten sind, dient eher der Bequemlichkeit des HTML-Autors. Es wird seitens des W3C jedoch angeraten, besser deren logische Pendants und zu verwenden.

HTML 4.01 strict enthält folgende 77 Elemente (auf die Endtags der Containerelemente wird in dieser Übersicht verzichtet – die Tagnamen werden hier als Unterscheidung zu XHTML in Großbuchstaben geschrieben, was für ihre Funktion jedoch irrelevant ist[9]):

<A> <ABBR> <ACRONYM> <ADDRESS> <AREA>
<B> <BASE> <BDO> <BIG> <BLOCKQUOTE> <BODY>   <BUTTON>
<CAPTION> <CITE> <CODE> <COL> <COLGROUP>
<DD> <DEL> <DFN> <DIV> <DL> <DT>
<EM> <FIELDSET> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> <IMG> <INPUT> <INS>
<KBD> <LABEL> <LEGEND> <LI> <LINK>
<MAP> <META>
<NOSCRIPT>
<OBJECT> <OL> <OPTGROUP> <OPTION>
<P> <PARAM> <PRE> <Q>
<SAMP> <SCRIPT> <SELECT> <SMALL> <SPAN> <STRONG> <STYLE> <SUB> <SUP>
<TABLE> <TBODY> <TD> <TEXTAREA> <TFOOT> <TH> <THEAD> <TITLE> <TR> <TT>
<UL> <VAR>

Tabelle 12.7 Elemente von HTML 4.01 strict

9 Ich rate generell zur Kleinschreibung von Tag- und Attributbezeichnern, unabhängig davon, ob HTML oder XHTML erzeugt werden soll.

12.3.2 HTML 4.01 transitional

Die *transitional*-Variante von HTML lässt mit Ausnahme der Frameset-elemente alle oben genannten Tags und Attribute weiterhin zu. Der Umfang des Tag-Vokabulars erweitert sich daher um zwölf (jeweils fett hervorgehoben) auf folgende 89 Elemente:

<A> <ABBR> <ACRONYM> <ADDRESS> **<APPLET>** <AREA>
<B> <BASE> **<BASEFONT>** <BDO> <BIG> <BLOCKQUOTE> <BODY>   <BUTTON>
<CAPTION> **<CENTER>** <CITE> <CODE> <COL> <COLGROUP>
<DD> <DEL> <DFN> **<DIR>** <DIV> <DL> <DT>
<EM> <FIELDSET> **<FONT>** <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> **<IFRAME>** <IMG> <INPUT> <INS> **<ISINDEX>**
<KBD> <LABEL> <LEGEND> <LI> <LINK>
<MAP> **<MENU>** <META>
<NOFRAMES> <NOSCRIPT>
<OBJECT> <OL> <OPTGROUP> <OPTION>
<P> <PARAM> <PRE> <Q>
<S> <SAMP> <SCRIPT> <SELECT> <SMALL> <SPAN> **<STRIKE>** <STRONG> <STYLE> <SUB> <SUP>
<TABLE> <TBODY> <TD> <TEXTAREA> <TFOOT> <TH> <THEAD> <TITLE> <TR> <TT>
<U> <UL> <VAR>

Tabelle 12.8 Elemente von HTML 4.01 transitional

12.3.3 HTML 4.01 frameset

Der Sprachumfang von *HTML 4.01 frameset* entspricht im Grunde der *transitional*-Variante und fügt dieser lediglich die beiden Frameset-Elemente <FRAMESET> und <FRAME> hinzu. (Interessanterweise ist der <NOFRAMES>-Container auch in *HTML 4.01 transitional* enthalten.) Der Sprachumfang wächst damit auf 91 Elemente:

<A> <ABBR> <ACRONYM> <ADDRESS> <APPLET> <AREA>
<B> <BASE> <BASEFONT> <BDO> <BIG> <BLOCKQUOTE> <BODY>   <BUTTON>
<CAPTION> <CENTER> <CITE> <CODE> <COL> <COLGROUP>
<DD> <DEL> <DFN> <DIR> <DIV> <DL> <DT>

Tabelle 12.9 Elemente von HTML 4.01 frameset

<EM> <FIELDSET> <FONT> <FORM> **<FRAME> <FRAMESET>**
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> <IFRAME> <IMG> <INPUT> <INS> <ISINDEX>
<KBD> <LABEL> <LEGEND> <LI> <LINK>
<MAP> <MENU> <META>
<NOFRAMES> <NOSCRIPT>
<OBJECT> <OL> <OPTGROUP> <OPTION>
<P> <PARAM> <PRE> <Q>
<S> <SAMP> <SCRIPT> <SELECT> <SMALL> <SPAN> <STRIKE> <STRONG> <STYLE> <SUB> <SUP>
<TABLE> <TBODY> <TD> <TEXTAREA> <TFOOT> <TH> <THEAD> <TITLE> <TR> <TT>
<U> <UL> <VAR>

Tabelle 12.9 Elemente von HTML 4.01 frameset (Forts.)

12.4 XHTML 1.0 und XHTML 1.1

In der Spezifikation zu XHTML 1.0 existieren, da es sich um eine exakte Reproduktion des Vokabulars von HTML 4.01 handelt, nach wie vor drei DTD-Varianten: XHTML 1.0 *strict*, *transitional* und *frameset*.

In XHTML 1.1 existiert nur noch eine Sprachvariante

Erst mit **XHTML 1.1** streift das W3C die missbilligten Sprachelemente vollständig ab: es existiert ausschließlich in einer *strict*-Variante.

Im Gegensatz zu HTML (wo entsprechende Unterscheidung zwischen Groß- und Kleinschreibung irrelevant war) hat man sich in XHTML für die **konsequente Kleinschreibung** der Tag- und Attributbezeichner entschieden, was in den folgenden Listen wiedergegeben ist.

Warum die Kleinschreibung von Bezeichnern in XHTML?

Eine Entscheidung für das eine oder das andere musste sein, da XML und deshalb auch XHTML »case-sensitive« sind, die Schreibweise der Bezeichner also relevant für die Gültigkeit des Dokuments ist.

Auch leere Elemente wie `<br>` oder `<hr>` müssen geschlossen werden – sie werden daher zur Verdeutlichung im Folgenden mit `<br />`, `<hr />` etc. gelistet. (Auf die Endtags der Containerelemente wird in dieser Übersicht dagegen wie zuvor verzichtet.)

12.4.1 XHTML 1.0 strict

Der Sprachumfang von *XHTML 1.0 strict* entspricht vollständig dem von *HTML 4.01 strict* und umfasst ebenfalls 77 Elemente:

<a> <abbr> <acronym> <address> <area />
<b> <base /> <bdo> <big> <blockquote> <body>   <button>
<caption> <cite> <code> <col> <colgroup>
<dd> <del> <dfn> <dl> <dt>
<fieldset> <form>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<i> <img /> <input /> <ins>
<kbd> <label> <legend>   <link />
<map> <meta />
<noscript>
<object>   <optgroup> <option>
<p> <param> <pre> <q>
<samp> <script> <select> <small>   <style> <sub> <sup>
<table> <tbody> <td> <textarea> <tfoot> <th> <thead> <title> <tr> <tt>
<var>

Tabelle 12.10 Elemente von XHTML 1.0 strict

12.4.2 XHTML 1.0 transitional

Der Sprachumfang von *XHTML 1.0 transitional* entspricht vollständig dem von *HTML 4.01 transitional* (89 Elemente; fett markiert: nicht in *strict* enthalten):

<a> <abbr> <acronym> <address> **<applet>** <area />
<b> <base /> **<basefont />** <bdo> <big> <blockquote> <body>   <button>
<caption> ** ** <cite> <code> <col> <colgroup>
<dd> <del> <dfn> **<dir>** <dl> <dt>
<fieldset> **<font>** <form>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<i> **<iframe>** <img /> <input /> <ins> **<isindex />**
<kbd> <label> <legend>   <link />
<map> **<menu>** <meta />
<noframes> <noscript>

Tabelle 12.11 Elemente von XHTML 1.0 transitional

<object>   <optgroup> <option>
<p> <param> <pre> <q>
<s> <samp> <script> <select> <small> **<strike>** <style> <sub> <sup>
<table> <tbody> <td> <textarea> <tfoot> <th> <thead> <title> <tr> <tt>
<u>   <var>

Tabelle 12.11 Elemente von XHTML 1.0 transitional (Forts.)

12.4.3 XHTML 1.0 frameset

Der Sprachumfang von *XHTML 1.0 frameset* entspricht vollständig dem von *HTML 4.01 frameset.* Gegenüber der *transitional*-Variante kommen die Element `<frame />` und `<frameset>` hinzu.

<a> <abbr> <acronym> <address> <applet> <area />
<b> <base /> <basefont /> <bdo> <big> <blockquote> <body>   <button>
<caption> <cite> <code> <col> <colgroup>
<dd> <del> <dfn> <dir> <dl> <dt>
<fieldset> <font> <form> **<frame /> <frameset>**
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<i> <iframe> <img /> <input /> <ins> <isindex />
<kbd> <label> <legend>   <link />
<map> <menu> <meta />
<noframes> <noscript>
<object>   <optgroup> <option>
<p> <param> <pre> <q>
<s> <samp> <script> <select> <small> <strike> <style> <sub> <sup>
<table> <tbody> <td> <textarea> <tfoot> <th> <thead> <title> <tr> <tt>
<u>   <var>

Tabelle 12.12 Elemente von XHTML 1.0 frameset

12.4.4 XHTML 1.1

Ab jetzt »echtes« XML: Der Versionssprung von XHTML 1.0 zu XHTML 1.1 bedeutet keine geringfügige Verbesserung, sondern einen echten Neuanfang. Zum einen sieht man von unmittelbarer Abwärtskompatibilität ab – es gibt keine *transitional*-Varianten mehr, sondern nur noch eine einzige DTD (die man

getrost auch als *strict* bezeichnen könnte). Zum anderen soll XHTML 1.1 als echtes XML-Format behandelt werden und von Server wie Browser auch entsprechend bearbeitet werden. Beispielsweise soll ein Webserver eine XHTML1.1-Datei mit dem **Mimetyp »application/xhtml+xml«** ausliefern[10], und dem Dokument soll ein XML-Prolog vorangestellt werden können.

In der Spezifikation zu XHTML 1.1 existieren im Gegensatz zur vorangegangenen Version 1.0 keine DTD-Varianten mehr. Tagsatz und Philosophie entspricht der *strict*-Kollektion von XHTML 1.0 mit einigen zusätzlichen Tags (hier fett hervorgehoben):

<a> <abbr> <acronym> <address> <area>
<b> <base /> <bdo> <big> <blockquote> <body>   <button>
<caption> <cite> <code> <col> <colgroup>
<dd> <del> <dfn> <dl> <dt>
<fieldset> <form>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<i> <img /> <input /> <ins>
<kbd> <label> <legend>   <link />
<map> <meta>
<noscript>
<object>   <optgroup> <option>
<p> <param> <pre> <q>
<ruby> <rb> <rbc> <rp> <rt> <rtc>
<samp> <script> <select> <small>   <style> <sub> <sup>
<table> <tbody> <td> <textarea> <tfoot> <th> <thead> <title> <tr> <tt>
<var>

Tabelle 12.13 Elemente von XHTML 1.1 strict

Die hinzugefügten sechs Elemente gehören dem **Ruby-Sprachmodul** an; ihre Aufgabe besteht darin, ein spezielles Textmarkup zu ermöglichen, das beispielsweise Besonderheiten in der Textnotation ostasiatischer Sprachen berücksichtigt. Eine mögliche Anwendung besteht in der Beifügung von Aussprachehinweisen oder Erklärungen.

10 Hierzu existiert eine W3C Note (1. August 2002) von Ishikawa Masayasu: »XHTML Media Types« siehe: www.w3.org/TR/xhtml-media-types.

Abbildung 12.6 Chinesisch mit Ruby-Annotationen

Was ist eine Ruby-Annotation?

Eine Ruby-Annotation ermöglicht ein quasi mehrzeiliges Textlayout mit einer Basiszeile und einer oder zwei darunter- bzw. darüberliegenden (erklärenden oder übersetzenden) Ruby-Textzeilen in kleinerer Schriftgröße. Je nach Schreibrichtung kann »darunter oder darüber« aber auch rechts oder links von Basistext bedeuten, beispielsweise bei vertikaler chinesischer Schreibrichtung.

Die Ruby-Spezifikation finden Sie unter **http://www.w3.org/TR/ruby/**. Es besteht kein Zusammenhang mit der Web 2.0 -Programmiersprache »Ruby«.

12.5 Bausteine: Die Modularisierung von XHTML

Das W3C hat die Sprachelemente von XHTML (sowohl die Deklarationen der Elemente als auch die ihrer Attribute) in eine Reihe abstrakter Module zerlegt, für die daher jeweils individuelle DTD- oder (seit XHTML Modularization 1.1) Schema-Fragmente existieren (im Falle der DTD als `.mod`-Dateien realisiert).

Ein Ziel dieser Maßnahme besteht darin, auf diesem Wege über eine Kombination dieser Module (fast) beliebige Varianten der **Reduktion** (*subset*) bzw. der **Erweiterung** (*extension*) des XHTML-Tagsatzes zu ermöglichen.

Hierfür können die Deklarations-Fragmente einzelner XHTML-Module miteinander, aber auch mit Deklarations-Fragmenten eigener Erweiterungsmodule kombiniert (zu so genannten »hybriden« Dokumenttypen) und gleichzeitig die Validierbarkeit entsprechend erzeugter Dokumente gewährleistet werden.

Des Weiteren stehen die XHTML-Module nun auch im Rahmen des »XHTML Modularization Frameworks« als Bausteine für beliebige andere XML-Anwendungen zur Verfügung.

Die Spezifikation »Modularization of XHTML«

Die aktuelle Version 1.1 der W3C-Spezifikation zur XHTML-Modularisierung finden Sie unter **http://www.w3.org/TR/xhtml-modularization** (letzter Stand 5. Juli 2006).

Es existieren 28 dieser Module, die allerdings nicht beliebig kombiniert werden können. Dies liegt daran, dass die Definitionen einiger Module auf denen anderer Module aufbauen (indem sie beispielsweise Elementen, die in anderen Modulen definiert sind, Attribute hinzufügen oder deren Inhaltsmodelle modifizieren). Bestimmte Module wie *Basic Table* und *Table* oder *Basic Forms* und *Forms* schließen einander aus, da die Basic-Variante jeweils eine (für bestimmte Verwendungszwecke abgespeckte) Untergruppe des anderen darstellt.

12.5.1 Der Common-Attributsatz

Es handelt sich hier nicht um ein eigenständiges Modul, sondern um die separate Definition vielfach eingesetzter Standardattribute. Aus Gründen der Rationalisierung werden solche häufig zugewiesenen Attributdefinitionen in verschiedenen Sammlungen (*collections*) zusammengefasst, die in ihrer Summe den allgemeinen Attributsatz (*common attribute set*) bilden.

Dieser stellt in fast allen Fällen den **Grundstock der Attributsammlung** eines XHTML-Elements dar, dem je nach Erfordernis weitere elementspezifische Attribute hinzugefügt sind.

Ausnahmen – Elemente ohne vollständigen Common-Attributsatz

Ausnahmen sind `<br />`, `<applet>`, `<bdo>`, `<map>`, `<iframe>`, `<frame />` und `<frameset>`, für die die Core-Attributsammlung als Basis gilt. Ebenfalls eingeschränkt sind `<head>`- und `<title>`-Container, der `<style>`-Container sowie `<meta />`, die nur die I18N-Sammlung erhalten. `<param>` erhält lediglich das `id`-Attribut, `<script>` und `<base />` keines der Standardattribute.

Zum Common-Attributsatz zählen folgende Sammlungen:

- **Core-Attributsammlung**
  enthält die drei Core-Attribute `id`, `title` und `class`
- **I18N-Attributsammlung**
  enthält stets das Sprachkennungs-Attribut `xml:lang` (im XML-Namensraum) und gegebenenfalls, sofern das Bi-Directional Text Module verwendet wird,

auch das `dir`-Attribut zur Kennzeichnung der Schreibrichtung eines Elementinhalts.

- **Events-Attributsammlung**
  enthält eine Reihe von Standard-Eventhandler-Attributen: `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`

[»] *Hinweis:* Die in XHTML für Attributnamen vorgeschriebene Kleinschreibung bedeutet zwangsläufig Abschied von der gewohnten Binnengroßschreibung. Statt z. B. `onMouseOver` *muss* nun stehen `onmouseover`.

- **Style-Attributsammlung**
  enthält das `style`-Attribut zur Definition von Inlinestyles, sofern das Style-Attribut Modul eingebunden ist

Anmerkung: Es ist wichtig, sich zu vergegenwärtigen, dass im Rahmen eines Moduls nicht die Attribute selbst einem Element zugewiesen werden, sondern die *Sammlungen* – dies kann gegebenenfalls einen Unterschied bedeuten.

Obwohl beispielsweise die Style-Attributsammlung Teil des Common-Attributsatzes ist und somit den meisten Tags zur Verfügung steht, bedeutet dies *nicht*, dass das `style`-Attribut selbst tatsächlich stets zugewiesen wird: Dies ist *nur dann* der Fall, wenn auch das Style-Attribut *Modul* im Sprachkontext verwendet wird, ansonsten wird die Style Attributsammlung einem Element zwar zugeordnet, ist aber leer. Analoges gilt für die Events-Attributsammlung, die das *Intrinsic Events Modul* voraussetzt.

Warum gibt es leere Attributsammlungen?

Die Gründe für dieses Manöver werden deutlich, wenn man sich vergegenwärtigt, dass sonst im Rahmen der Style-Attribut- bzw. Intrinsic Events Module explizit definiert werden müsste, für welchen einzelnen Tag die Zuweisung gelten soll.

12.5.2 Core-Module

Vier der im nachfolgenden vorgestellten Module bilden den **Sprachkern** (*core*) von XHTML. Ihre Anwesenheit in einer Modulkombination ist erforderlich, damit die so gebildete Sprache als zur XHTML-Familie zugehörig betrachtet wird (dieses Kriterium wird als »komform zu XHTML als *host language*« bezeichnet; s. u.). Es handelt sich hierbei um

- Das Strukturmodul (*structure module*)
- Das Textmodul (*text module*)

- Das Hypertextmodul (*hypertext module*)
- Das Listenmodul (*list module*)

In diesen vier Modulen werden grundlegende Elemente von XHTML, deren Inhaltsmodelle und Attribute definiert.

Konformität der Modulkombinationen

In Bezug auf die Dokumenttypen, die durch Kombination eigener Module mit XHTML-Modulen erzeugt werden, unterscheidet man verschiedene Ausprägungen der Konformität:

- konform zum **XHTML Integration Set** Dokumenttyp

  Eine Sprache gilt als konform zum XHTML Integration Set, wenn sie (mindestens) die XHTML-Module für Text, Hypertext und Listen, *nicht aber das Strukturmodul* einbeziehen (eine solche Sprache braucht daher u.a. nicht das Element `<html>` als Wurzelelement zu verwenden).

  Der Name der DTD bzw. des Schemas der so erzeugten Sprache *darf nicht* mit dem Wort XHTML beginnen. Eine solche Sprache gehört per Definition *nicht* zur XHTML-Familie.

  Namensbeispiel: `-//MeineFirma//DTD Spezial Markup mit XHTML//DE`

- konform zum **XHTML Host Language** Dokumenttyp

  Eine Sprache bedient sich XHTML als »Wirtssprache« (*host language*), wenn sie in vollem Umfang neben den XHTML-Modulen für Text, Hypertext und Listen auch[11] das **Strukturmodul** übernimmt (eine solche Sprache wird daher das Wurzelelement `<html>` besitzen). Darüber hinaus können weitere XHTML-Module einbezogen werden, aber auch eigene Erweiterungen (mit eigenem Namensraum).

  Der Name der DTD bzw. des Schemas der so erzeugten Sprache *muss* mit dem Wort XHTML beginnen. Eine solche Sprache gehört per Definition zur **XHTML-Familie**, beispielsweise »XHTML Basic« oder »XHTML mobile profile«.

  Namensbeispiel: `-//MeineFirma//DTD XHTML Spezial 1.0//DE`

11 Eine solche Kombination ist automatisch, quasi als ein diesem übergeordneten Konzept ebenfalls konform zum XHTML-Integration Set.

12.5.3 Das Strukturmodul

Das XHTML-Strukturmodul zählt zu den vier Core-Modulen. Seine Implementierung innerhalb einer mit Hilfe von XHTML-Modulen gebildeten Markup-Sprache ist unabdingbar, wenn diese zur XHTML-Familie gezählt werden soll.

Das Modul enthält die vier grundlegenden Elemente, die das **Grundgerüst** jedes HTML-Dokuments ausmachen:

Element	Rolle	Typ
`<html> ... </html>`	Wurzelelement des Dokuments	Strukturcontainer
`<head> ... </head>`	Dokument*kopf*; enthält neben dem Titel weitere Metainformationen, Stilanweisungen oder Scriptblöcke	Strukturcontainer
`<title> ... </title>`	Dokument*titel*; Teil des Dokumentkopfes	Strukturcontainer
`<body> ... </body>`	Dokument*körper*; enthält den sichtbaren Teil des HTML-Dokuments	Strukturcontainer

Tabelle 12.14 Elemente des Strukturmoduls

12.5.4 Das Textmodul

Auch das Textmodul gehört zu den Core-Modulen. Die Elemente, die dieses Modul umfasst, dienen der **semantischen Textauszeichnung**, besitzen also per se keine festgelegten Präsentationseigenschaften. Sie bezeichnen die logische Bedeutung des eingeschlossenen Inhalts, sagen jedoch nichts Verbindliches darüber aus, wie dieser dargestellt werden soll.

Warum sind die Elemente <b> und <i> nicht semantisch?

Die Tags `<b>` und `<i>` bezeichnen im Gegensatz zu `<strong>` und `<em>` die Präsentation als **fett** oder *kursiv*, ohne über die Bedeutung ihres Inhaltes etwas auszusagen. Sie werden deshalb nicht dem Text-, sondern dem Präsentationsmodul zugeschlagen. Auch für die Elemente der semantischen Textauszeichnungen existieren Darstellungskonventionen – ein Element dieser Gruppe aus diesem Grunde zu verwenden (z. B. `<blockquote>` als Einrückung), ist jedoch streng genommen Missbrauch oder zumindest ein Missverständnis.

Die folgende Tabelle enthält eine Übersicht über die Elemente des **Textmoduls**. Die Unterscheidung nach Block- oder Inlinetyp geschieht nach dem Defaultverhalten, das jedoch durch Stylesheetanweisungen widerrufen werden kann.

Element	Rolle	Typ
`<abbr> ... </abbr>`	Logische Auszeichnung des Inhalts als Abkürzung	Inlinecontainer
`<acronym> ... </acronym>`	Auszeichnung als Akronym[12]	Inlinecontainer
`<address> ... </address>`	Auszeichnung als Adresse; gewöhnlich kursiv und evtl. eingerückt präsentiert	Blockcontainer
`<blockquote>... </blockquote>`	Auszeichnung als Zitatblock; gewöhnlich eingerückt präsentiert	Blockcontainer
` `	Erzwungener Zeilenumbruch; wird zu den Blockelementen gezählt	Leeres Element; Blockcharakter
`<cite> ... </cite>`	Bezeichnet die Quelle (Herkunft) eines Zitats	Inlinecontainer
`<code> ... </code>`	Bezeichnet Quellcode eines Programmabschnitts; gewöhnlich mit nicht-proportionaler Schrift konstanter Breite (Courier o.ä.) dargestellt	Inlinecontainer
`<dfn> ... </dfn>`	Bezeichnet eine Definition; gewöhnlich kursiv dargestellt	Inlinecontainer
`<div> ... </div>`	Bezeichnet einen logischen Block, Abschnitt oder Bereich	Blockcontainer
`<em> ... </em>`	Logisches Pendant zu `<i></i>`; bezeichnet einen hervorgehobenen Bereich (*emphasized* = betont); wird gewöhnlich kursiv dargestellt	Inlinecontainer
`<h1> ... </h1>` `<h2> ... </h2>` `<h3> ... </h3>` `<h4> ... </h4>` `<h5> ... </h5>` `<h6> ... </h6>`	Überschriften in von `<h1>` bis `<h6>` fallender Hierarchie; gewöhnlich fett und mit Abstand zum folgenden Abschnitt dargestellt; Schriftgrößen spiegeln von groß nach klein die Hierarchiestufen 1 bis 6 wider	Blockcontainer
`<kbd> ... </kbd>`	Bezeichnet eine Tastatureingabe; übliche Darstellung wie `<code>`	Inlinecontainer
`<p> ... </p>`	Bezeichnet einen Textabsatz	Blockcontainer
`<pre> ... </pre>`	Bezeichnet einen Bereich, für den Leerzeichen, Umbrüche und Tabulatoren des Quelltextes[13] für die Präsentation übernommen werden sollen; Darstellung wie `<code>`	Blockcontainer

Tabelle 12.15 Elemente des Textmoduls

12 Definition lt. Duden (Mannheim 2005): »Kunstwort, das aus den Anfangsbuchstaben mehrerer Wörter zusammengesetzt ist.«

13 Technisch gesehen setzt `<pre>` das reguläre Parsingverhalten des Webbrowsers gegenüber Whitespace außer Kraft. Es ist Ansichtssache, ob es sich hier nicht eigentlich um einen Präsentationstag handelt, da keine semantische Aussage über den Inhalt getroffen wird.

Element	Rolle	Typ
`<q> ... </q>`	Bezeichnet einen zitierten Abschnitt innerhalb eines Textblocks (Inlinezitat); Kennzeichnung durch Umgeben mit Anführungszeichen	Inlinecontainer
`<samp> ... </samp>`	"sample"; bezeichnet einen Auszug eines längeren Abschnitts einer Quelle (z. B. Programmquellcode);	Inlinecontainer
`<span> ... </span>`	Neutraler Container; wichtig in Zusammenhang mit Stylesheets	Inlinecontainer
`<strong> ... </strong>`	Logisches Pendant zu `<b></b>`; bezeichnet einen stark hervorgehobenen Bereich; wird gewöhnlich fett dargestellt	Inlinecontainer
`<var> ... </var>`	Bezeichnet eine Variable	Inlinecontainer

Tabelle 12.15 Elemente des Textmoduls (Forts.)

Alle Elemente des Textmoduls besitzen den Common-Attributsatz. Die Elemente `<blockquote>` und `<q>` erhalten zusätzlich das `cite`-Attribut, `<pre>` das Whitespace-Attribut `xml:space` mit festem Wert `"preserve"`. Alle Präsentationsattribute für Textausrichtung etc. entfallen, stehen aber als Teil des Legacy Moduls zur Verfügung.

12.5.5 Das Hypertextmodul

Das Hypertextmodul gehört zu den Core-Modulen, da die Möglichkeit der Definition von Hyperlinks elementar für XHTML ist. Es umfasst als einziges Element den Ankertag `<a>` und einige seiner Attribute.

Element	Rolle	Typ
`<a> ... </a>`	"anchor"; bezeichnet Quelle oder Ziel eines Hyperlinks	Inlinecontainer

Tabelle 12.16 Elemente des Hypertextmoduls

Folgende Attribute werden zusammen mit dem Element `<a>` im Rahmen des Moduls deklariert (der Common-Attributsatz steht ebenfalls zur Verfügung):

`accesskey`, `charset`, `href`, `hreflang`, `rel`, `rev`, `tabindex`, `type`

Anmerkung: Beachtenswert ist, welche Attribute Teil dieses Moduls sind und welche nicht: Es *fehlen* die (vielleicht an dieser Stelle erwarteten) Attribute `name` und `target`.

Das name-Attribut entfällt zugunsten des id-Attributs (Letzteres als Core-Attribut mit der allgemeinen Attributsammlung auch dem <a>-Tag zugewiesen), steht aber im Legacy Modul zusammen mit anderen »überkommenen« Sprachelementen zur Verfügung.

Das target-Attribut ist ausgelagert, da es meist nur im Framesetkontext benötigt wird. Dieses ist jedoch nicht Teil des Frame Moduls, sondern stellt als Target Modul einen eigenen Baustein für sich dar (siehe dort).

12.5.6 Das Listenmodul

Das Listenmodul gehört ebenfalls zu den vier Core-Modulen. Es enthält die Listentypen für geordnete und ungeordnete Listen, Definitionslisten und deren jeweils zugehörige Listenelemente.

Element	Rolle	Typ
 ... 	"unordered list"; Container für eine ungeordnete Liste; Listenelemente werden durch Bullets gekennzeichnet	Blockcontainer
 ... 	"ordered list"; Container für eine geordnete Liste; Listenelemente erhalten Nummerierung mit Ziffern oder Buchstaben	Blockcontainer
...	"list item"; Listenelement einer ungeordneten oder geordneten Liste; Darstellung abhängig von der Art des umgebenden Listencontaners	Blockcontainer
<dl> ... </dl>	"definition list"; Container für eine Definitionsliste; enthält abwechselnd <dt> und <dl>-Container	Blockcontainer
<dt> ... </dt>	"definition term"; bezeichnet den zu definierenden Ausdruck einer Definition	Blockcontainer
<dd> ... </dd>	"definition data"; bezeichnet die zum vorstehenden <dt>-Container gehörende Definition	Blockcontainer

Tabelle 12.17 Elemente des Listenmoduls

Anmerkung: Alle Elemente des Listenmoduls besitzen ausschließlich den Common-Attributsatz. Reine Präsentationsattribute, wie compact, start, type fehlen; es wird empfohlen, deren Aufgaben durch Styleanweisungen vorzunehmen. Die Attribute stehen dennoch als Teil des Legacy Moduls zur Verfügung.

Auch die Elemente <dir> (*directory list*) und <menu> (*menu list*), die in HTML ebenfalls Listencharakter besitzen, gehören nicht zu diesem Modul, sondern sind, da sie als »deprecated« gelten, ebenfalls Teil des Legacy Moduls.

Hat das Menülistenelement doch eine Zukunft?

XHTML 2.0 führt interessanterweise wieder ein Navigationslistenelement <nl> ein, das dann Teil des (dort erweiterten) Listenmoduls ist. Da die Spezifikation noch immer Draft-Charakter hat (Stand Februar 2007), mag sich hier jedoch noch einiges ändern. Die WHAT WG will stattdessen <menu> beibehalten.

12.5.7 Das Präsentationsmodul (presentation module)

Zur Erzeugung von orthodoxem XHTML ist dieses Modul im Grunde fragwürdig, da die in ihm enthaltenen Elemente, wie auch der Name des Moduls unterstreicht, lediglich Präsentationscharakter besitzen.

Was bedeutet Präsentationscharakter?

All diese Elemente können zumindest theoretisch durch + CSS oder semantische Elemente wie , oder <code> etc. ersetzt werden oder im Falle von <hr /> durch Styleanweisungen für die border-Properties benachbarter Container.

Dies gilt auch für <sub> und <sup>, die zwar zur Präsentation von Massenverhältniszahlen in chemischen Verbindungen oder zur Darstellung mathematischer Exponenten verwendet werden, deren semantischen Aspekt jedoch nicht beinhalten. Aus orthodoxer Sicht wäre es korrekter, hierfür Auszeichnungselemente aus MathML oder ChemML zu verwenden.

Die im **Präsentationsmodul** zusammengestellten Elemente haben sich jedoch aufgrund ihrer einfachen Verwendbarkeit so stark etabliert, dass sie nicht dem Legacy Modul zugeordnet wurden.

Element	Rolle	Typ
...	"bold"; präsentiert den Containerinhalt in Fettauszeichnung; Alternative:	Inlinecontainer
<big> ... </big>	Präsentiert den Containerinhalt relativ zum umgebenden Container[14] in größerer Schrift (ca. 120 %)	Inlinecontainer

Tabelle 12.18 Elemente des Präsentationsmoduls

14 Jeder <big>-Container erhöht die Schriftgröße jeweils um eine Stufe, zwei verschachtelte <big>-Container dementsprechend um zwei Stufen usw. Analoges gilt für <small>.

Element	Rolle	Typ
`<hr />`	"horizontal rule"; fügt eine Trennung zwischen zwei Abschnitten ein; gewöhnlich als horizontale Linie realisiert	Leeres Element; Blockcharakter
`<i> ... </i>`	"italic"; präsentiert den Containerinhalt in Kursivauszeichnung Alternative: `<em></em>`	Inlinecontainer
`<small> ... </small>`	Präsentiert den Containerinhalt relativ zum umgebenden Container in kleinerer Schrift (ca. 80 %)	Inlinecontainer
`<sub> ... </sub>`	"subscript"; Inhalt wird tiefergestellt und eine Schriftgröße kleiner (ca. 80 %) dargestellt	Inlinecontainer
`<sup> ... </sup>`	"superscript"; Inhalt wird hochgestellt und eine Schriftgröße kleiner (ca. 80 %) dargestellt	Inlinecontainer
`<tt> ... </tt>`	"teletype"; Inhalt wird mit nicht-proportionaler Schrift konstanter Breite (Courier o.ä.) dargestellt. Alternativen (je nach Kontext): `<code></code>`, `<samp></samp>`, `<keyb></keyb>`	Inlinecontainer

Tabelle 12.18 Elemente des Präsentationsmoduls (Forts.)

Anmerkung: Das Element `<hr />` stellt in diesem Zusammenhang das einzige Blockelement dar. Da es oft in dekorativer Funktion verwendet wird, ist die Zuordnung zu diesem Modul gerechtfertigt. In vielen, jedoch nicht allen Fällen kann die optische Trennung jedoch durch Border-Styleanweisungen für den benachbarten Container ersetzt werden.

In **XHTML 2.0** wird diese Problematik berücksichtigt, indem der `<hr />`-Tag durch das (semantisch zu verstehende) `<separator>`-Element ersetzt wird. Dessen Default-Präsentation ist ebenfalls ein horizontaler Trennstrich.

12.5.8 Das Formular-Basismodul (Basic Forms Module)

Das Formular-Basismodul enthält in Form der sechs entsprechenden Elemente das minimal erforderliche Vokabular für den Einsatz von Formularen. Es wird beispielsweise in XHTML Basic eingesetzt.

Element	Rolle	Typ
`<form> ... </form>`	Bildet einen Container für Formularelemente; bestimmt über Attribute das Formularverhalten	Blockcontainer
`<input />`	Erzeugt verschiedene Arten von Inputelementen, deren Form über das `type`-Attribut bestimmt wird	Leeres Element; Inlinecharakter
`<label> ... </label>`	Erzeugt zu einem Formularelement über eine IDREF-ID-Beziehung eine klickbare Beschriftung	Inlinecontainer
`<select> ... </select>`	Selectbox; dient als Container für Optionselemente	Inlinecontainer
`<option> ... </option>`	Erzeugt als Kindelement des Select-Containers dessen einzelne Wahloptionen	Container
`<textarea> ... </textarea>`	Erzeugt ein Texteingabefeld; über Attribute werden dessen Abmessungen festgelegt	Inlinecontainer

Tabelle 12.19 Elemente des Basic Forms Moduls

Mit Hilfe des **Formular-Basismoduls** lassen sich einfache, aber voll funktionstüchtige Formulare erzeugen. Einige Elemente wie z. B. `<fieldset>` u. a. fehlen hier zugunsten besserer Übersichtlichkeit und Vereinfachung. Als **Inputtypen** stehen `"text"`, `"password"`, `"checkbox"`, `"radio"`, `"submit"`, `"reset"` und `"hidden"` zur Verfügung, *nicht* aber `"file"` und `"image"`.

Weniger offensichtlich ist, dass zum Teil[15] auch in Bezug auf die Attribute der im Modul enthaltenen Elemente bzw. deren erlaubte Werte abgespeckt wurde. Die Attribute bzw. Attributwerte der rechten Spalte der folgenden Tabelle stehen in Basic Forms also *nicht* zur Verfügung. Werden sie benötigt, muss stattdessen das Forms Modul eingebunden werden:

Element	Attribute in »Basic Forms«	zusätzl. Attribute in »Forms«

`<form>`	*Common*
	`action`
	`method`
	`enctype`
		+ `accept`
		+ `accept-charset`

Tabelle 12.20 Vergleich der Attribute der Elemente des Basic Forms und des Forms Moduls

15 In der folgenden Tabelle *nicht* aufgeführte Elemente besitzen im Forms Modul die gleiche Attributsammlung wie im Basic Forms Modul.

Element	Attribute in »Basic Forms«	zusätzl. Attribute in »Forms«
<input />	*Common* accesskey checked maxlength name size src tabindex type value	+ alt + disabled + readonly + type="button" + type="file" + type="image"
<select>	*Common* multiple name size tabindex	+ disabled
<option>	*Common* selected value	+ disabled
<textarea>	*Common* accesskey cols name tabindex	+ disabled + readonly

Tabelle 12.20 Vergleich der Attribute der Elemente des Basic Forms und des Forms Moduls (Forts.)

Anmerkung: Beachtenswert ist, dass das `name`-Attribut, das ansonsten zugunsten von `id` fallen gelassen wird, hier für alle **Eingabeelemente** erhalten bleibt (Vorsicht: nicht jedoch für `<form>` selbst!). Das `id`-Attribut steht über die Common-Attributsammlung ebenfalls zur Verfügung. Wird in Zusammenhang mit Elementen anderer Module das `name`-Attribut benötigt, so kann es über das (missbilligte) *Name Identification Module* geladen werden (es ist nicht Teil des Legacy Moduls).

12.5.9 Tabellen-Basismodul (Basic Tables Module)

Das Tabellen-Basismodul enthält ein aus fünf Elementen bestehendes Minimalvokabular für die Erstellung von Tabellen. Das Modul wird beispielsweise in XHTML Basic eingesetzt.

Element	Rolle	Typ

`<table>`
`...`
`</table>`
	Definiert eine Tabelle	Blockcontainer

`<caption> ...` `</caption>`	Erläuternde Überschrift über (Default) oder unter der Tabelle; erstes Unterelement von `<table>`	Blockcontainer

`<tr>`
`...`
`</tr>`
	Definiert eine Tabellenzeile als Unterelement von `<table>`	Container

`<td> ... </td>`	Definiert eine Tabellenzelle als Unterelement von `<tr>`	Container
`<th> ... </th>`	Definiert eine Tabellenkopfzelle als Unterelement von `<tr>`	Container

Tabelle 12.21 Elemente des Basic Table Moduls

Mit Hilfe des **Tabellen-Basismoduls** lassen sich einfache Tabellen erstellen. Gegenüber dem vollständigen Tabellenmodul wurde auf Gruppierungscontainer wie `<col>`, `<colgroup>`, `<thead>`, `<tbody>`, `<tfoot>` verzichtet.

Alle Elemente verfügen über die Common-Attributsammlung und weitere tagspezifische Attribute. Auch hier ist der Attributsatz der im Basismodul enthaltenen Elemente bzw. deren erlaubten Werte zum Teil gegenüber dem vollständigen Tabellenmodul reduziert.

Die Attribute bzw. Attributwerte der rechten Spalte der folgenden Tabelle stehen in Basic Tables also *nicht* zur Verfügung. Werden sie benötigt, muss stattdessen das Tables Modul eingebunden werden.

Element	Attribute in »Basic Tables«	zusätzl. Attribute in »Tables«

`<table>`	*Common*
	`summary`
	`width`
		+ `border`,
		+ `cellpadding`
		+ `cellspacing`
		+ `datapagesize`
		+ `frame`
		+ `rules`

`<tr>`	*Common*
	`align="left"`
	`align="center"`
	`align="right"`
	`valign="top"`
	`valign="bottom"`
		+ `char`,
		+ `charoff`
		+ `align="char"`
		+ `align="justify"`
		+ `valign="baseline"`

Tabelle 12.22 Vergleich der Attribute der Elemente des Basic Table und des Table Moduls

Element	Attribute in »Basic Tables«	zusätzl. Attribute in »Tables«
<td>, <th>	*Common* abbr axis colspan rowspan headers scope align valign	+ abbr + char + charoff, + scope="rowgroup" + scope="colgroup" + align="char" + align="justify" + valign="baseline"

Tabelle 12.22 Vergleich der Attribute der Elemente des Basic Table und des Table Moduls (Forts.)

Anmerkung: Beachtenswert ist hier, dass die Ausrichtung von Inhalten in Zellen bzw. Zeilen nach wie vor über `align`- und `valign`-Attribute erfolgen darf (Werte `"left"`, `"center"`, `"right"`, respektive `"top"`, `"middle"`, `"bottom"`).

Dem `<table>`-Container fehlen das Attribut für `border` sowie diejenigen für `cellpadding` und `cellspacing`, was bei letzteren deren **Defaultwerte** ungleich Null vermutlich nicht außer Kraft setzt. Für `cellspacing` liegt der Wert (browserabhängig) meist bei 1 px, für `cellpadding` bei 2 px. Hier ergibt sich das Problem, dass `cellspacing` über CSS derzeit browserübergreifend nur mit Schwierigkeiten zu erfassen ist.

12.5.10 Formularmodul

Das Formularmodul enthält den vollständigen Satz aller zehn in HTML 4.01 und XHTML 1.0 zur Erstellung von Formularen definierten Elemente und deren Attribute.

Element	Rolle	Typ
<button> </button>	Definiert einen Formularbutton; Containerinhalt wird auf Klickfläche dargestellt	Inlinecontainer
<fieldset> ... </fieldset>	Definiert eine zusammengehörende Gruppe von Inputelementen eines Formulars	Blockcontainer
<form> ... </form>	Bildet einen Container für Formularelemente; bestimmt über Attribute das Formularverhalten	Blockcontainer
<input />	Erzeugt verschiedene Arten von Inputelementen, deren Form über das `type`-Attribut bestimmt wird	Leeres Element; Inlineverhalten

Tabelle 12.23 Elemente des Forms Moduls

Element	Rolle	Typ
<label> ... <label>	Erzeugt zu einem Formularelement über eine IDREF-ID-Beziehung eine klickbare Beschriftung	Inlinecontainer
<legend> ... </legend>	Definiert eine Überschrift einer Fieldsetgruppe	Inlinecontainer
<optgroup> ... </optgroup>	Definiert eine zusammengehörende Gruppe von Optionselementen einer Selectbox	Container
<option> ... </option>	Erzeugt als Kindelement des Select-Containers dessen einzelne Wahloptionen	Container
<select> ... </select>	Selectbox; dient als Container für Optionselemente	Inlinecontainer
<textarea> ... </textarea>	Erzeugt ein Texteingabefeld; über Attribute werden dessen Abmessungen festgelegt	Inlinecontainer

Tabelle 12.23 Elemente des Forms Moduls (Forts.)

Das komplette **Formularmodul** enthält alle Elemente des **Formular-Basismoduls**, erweitert jedoch deren Attributsätze (auch hier gilt nach wie vor das name-Attribut) und enthält weitere Elemente zur Gruppierung oder Beschriftung.

Als **Inputtypen** steht der komplette Satz zur Verfügung, also "text", "password", "checkbox", "radio", "submit", "reset", "file", "image" und "hidden". Als weiterer Inputtyp ist das Element <button> zugelassen, das Kindelemente und Text als Containerinhalt besitzen darf, der dann die Buttonoberfläche bildet.

12.5.11 Das Tabellenmodul

Im **Tabellenmodul** enthalten sind alle zehn in XHTML bzw. HTML definierten Elemente, die den Aufbau einfacher oder komplexer Tabellen ermöglichen.

Element	Rolle	Typ
<table> ... </table>	Definiert eine Tabelle	Blockcontainer
<caption> ... </caption>	Erläuternde Überschrift über (Default) bzw. unter der Tabelle; erstes Unterelement von <table>	Blockcontainer
<tr> ... </tr>	Definiert eine Tabellenzeile als Unterelement von <table>	Container

Tabelle 12.24 Elemente des Table Moduls

Element	Rolle	Typ
`<th> ... </th>`	Definiert eine Tabellenkopfzelle als Unterelement von `<tr>`	Container
`<td> ... </td>`	Definiert eine Tabellenzelle als Unterelement von `<tr>`	Container
`<colgroup> </colgroup>`	Gruppiert mit dem `<col>`-Tag vorgenommene Spaltendefinitionen	Container; Inhalt wird nicht dargestellt
`<col />`	Definiert die Eigenschaften einer Tabellenspalte als Teil einer Spaltengruppe (`<colgroup>`)	Leeres Element
`<thead> ... </thead>`	Erklärt die Tabellenzeile(n), die der Container enthält, zum Tabellenkopf	Container
`<tfoot> ... </tfoot>`	Erklärt die Tabellenzeile(n), die der Container enthält, zum Tabellenfuß	Container
`<tbody> ... </tbody>`	Erklärt die Tabellenzeile(n), die der Container enthält, zum Tabellenkörper	Container

Tabelle 12.24 Elemente des Table Moduls (Forts.)

Anmerkung: Die Tabellenbereichscontainer `<thead>` und `<tfoot>` werden zwar von den meisten aktuellen Browsern unterstützt, was sich jedoch nicht immer auch auf die Druckausgabe erstreckt (nach der bei einer sich über mehrere Druckseiten erstreckenden Tabelle der `<thead>` am Anfang jeder Druckseite wiederholt werden soll; Analoges gilt für `<tfoot>`).

12.5.12 Das Image Modul

Das **Image Modul** enthält als einziges Element den `<img>`-Tag und seine Attribute.

Element	Rolle	Typ
`<img />`	Bindet eine Grafik in eine Webseite ein	Leeres Element; Inlineverhalten

Tabelle 12.25 Elemente des Image Moduls

Als Attributsammlung stehen für `<img>` neben der Common Collection die optionalen Attribute `height`, `width` und `longdescr` sowie die beiden obligatorischen Attribute `alt` und `src` zur Verfügung.

12.5.13 Das Applet Modul

Das **Applet Modul** gilt zugunsten des alternativ einsetzbaren **Object Moduls** als »missbilligt«. Die beiden im Rahmen dieses Moduls definierten Elemente dienen zur Einbindung von Anwendungen in HTML-Seiten (in der Regel wird es sich um Java-Applets handeln).

Element	Rolle	Typ
`<applet> ... </applet>`	Bindet eine Applikation in eine Webseite ein	Inlinecontainer
`<param />`	Übergibt Parameterwerte an den umgebenden `<applet>`-Container	Leeres Element

Tabelle 12.26 Elemente des Applet Moduls

Anmerkung: Das `<param>`-Element ist ebenfalls Teil des **Object Moduls**. In diesem Kontext dient es als Inhalt des `<object>`-Containers, dem in diesem Fall die Parameterwerte übergeben werden.

12.5.14 Das Base Modul

Das **Base Modul** enthält ein Element `<base>`, das einen Basis-URI definiert, gegen den relative URIs im Inneren des Dokumentes aufgelöst werden.

Element	Rolle	Typ
`<base />`	Definiert den Basis-URI des Dokuments	Leeres Element

Tabelle 12.27 Elemente des Base Moduls

Anmerkung: Das Element erhält ein `href`-Attribut (*nicht* jedoch den Common-Attributsatz). Ein `target`-Attribut wird `<base>` dann zusätzlich zugeordnet, wenn mit dem **Base Modul** gleichzeitig das Target Modul eingebunden wird (in der Regel im Framesetkontext).

12.5.15 Das Bi-Directional Text Modul

Das **Bi-Directional Text Modul** bietet die Möglichkeiten, die Textleserichtung innerhalb eines Dokuments oder Teilen davon zu definieren. Das Modul enthält zum einen das `<bdo>`-Element, das mittels seines `dir`-Attributs die Leserichtung des eingeschlossenen Textbereiches bestimmt. Gleichzeitig wird das `dir`-Attribut auch der **I18N-Attributsammlung** hinzugefügt und steht damit als Teil der Common-Attributsammlung allen Elementen zur Verfügung, denen diese Attributsammlung zugeordnet ist.

Element	Rolle	Typ
`<bdo> ... </bdo>`	Definiert die Leserichtung `rtl` ("right to left") oder `ltr` ("left to right")	Inlinecontainer

Tabelle 12.28 Elemente des Bi-Directional Text Moduls

12.5.16 Das Client-Side Image Map Modul

Das Client-Side Image Map Modul beinhaltet die Elemente, die zur Erzeugung von Imagemaps im Browser erforderlich sind. Die Verwendung dieses Moduls setzt die gleichzeitige Einbindung des Image Moduls (oder des Object Moduls) *zwingend* voraus.

Element	Rolle	Typ
`<area />`	Definiert Koordinaten und Umriss eines klickbaren Bereichs	Leeres Element; unsichtbar
`<map> ... </map>`	Gruppiert `<area/>`-Elemente zu einer Imagemapdefinition	Container; unsichtbar

Tabelle 12.29 Elemente des Client-Side Image Map Moduls

Das `<map>`-Element erhält nicht den vollständige Common-Attributsatz, sondern lediglich `xml:lang`, `class`, `id` und `title` sowie die Events Collection (die Style-Collection ist in der Tat überflüssig, da das Element nicht sichtbar ist).

Das `<area>`-Element erhält neben der Common Collection als weitere Attribute `accesskey`, `tabindex`, `coords`, `alt`, `href`, `nohref` (mit Wert `"nohref"`), `shape` (mit den Werten `"rect"`, `"circle"`, `"poly"` oder `"default"`).

Einigen in anderen Modulen definierten Elementen werden weitere Attribute hinzugefügt. Betroffen sind der `<a>`-Tag (aus dem Hyperlink Modul, stets eingebunden) sowie die Elemente `<img />` und `<object>`.

Attribut	Rolle	erweitert den Attributsatz für
`coords`	Bestimmt die Koordinaten des klickbaren Bereichs	`<a>`
`shape`	Bestimmt die Form eines klickbaren Bereichs; erlaubte Werte: `"rect"`, `"circle"`, `"poly"`, `"default"`	`<a>`
`usemap`	Bestimmt den Bezeichner der Imagemap, die für ein Bild oder Objekt zu verwenden ist	`<img />` `<object>`

Tabelle 12.30 Attributadditionen durch das Client-Side Image Map Modul

Anmerkung: Je nachdem, ob Bilder oder Objekte als Imagemaps dienen sollen, müssen alternativ oder gleichzeitig **Image Modul** oder **Object Modul** eingebunden sein.

12.5.17 Das Edit Modul

Dieses Modul enthält zwei Elemente, die vorgenommene **Textbearbeitungen**, also Streichungen und Hinzufügungen, verdeutlichen können. (Es existiert kein eigenes Element für »allgemeine Änderungen« unter der Prämisse, dass jede Änderung entweder eine Streichung oder eine Hinzufügung ist oder sich aus einer Kombination dieser beiden zusammensetzt.)

Element	Rolle	Typ
`<ins> ... </ins>`	"inserted"; markiert einen Textabschnitt als hinzugefügt	Inlinecontainer
`<del> ... </del>`	"deleted"; markiert einen Textabschnitt als gestrichen	Inlinecontainer

Tabelle 12.31 Elemente des Edit Moduls

Neben dem Common-Attributsatz besitzen beide Elemente ein `cite`- und ein `datetime`-Attribut, um Urheber/Quelle und Zeitpunkt der Änderung zu dokumentieren.

Anmerkung: Es ist nicht festgelegt, auf welche Art und Weise Textbearbeitungen im Browser vorgenommen oder dargestellt werden sollen. Derzeit werden diese Elemente oft nicht unterstützt bzw. sind mit keinem Default-Präsentationsstil versehen. Für `<del>` läge `display:none` oder `text-decoration:line-through` nahe, für `<ins>` eine wie auch immer geartete Farbgestaltung als Highlighting.

Auch das W3C markiert daher für die **Diff-Versionen** seiner Spezifikationen Änderungen noch mit dem `<span>`-Container und speziellen CSS-Klassen, unterscheidet hier aber **drei Zustände** (alt/gestrichen, geändert, neu/hinzugefügt).

12.5.18 Das Frames Modul

Das Frames Modul enthält die drei Elemente, die zur Erstellung von Framesets oder zu ihnen alternativen Inhalten erforderlich sind.

Element	Rolle	Typ
`<frame />`	Definiert einen Frame innerhalb eines Framesets	Leeres Element

Tabelle 12.32 Elemente des Frames Moduls

Element	Rolle	Typ
`<frameset> ... </frameset>`	Gruppiert Frame- und Framesetdefinitionen zu einem Frameset	Container
`<noframes> ... </noframes>`	Enthält alternativen Content zum Frameset	Container

Tabelle 12.32 Elemente des Frames Moduls (Forts.)

Anmerkung: Das Frames Modul ändert das DTD-Inhaltsmodell des `<html>`-Containers zu `(head, frameset)` und setzt die gleichzeitige Einbindung des **Target Moduls** zwingend voraus.

12.5.19 Das iFrame Modul

Das iFrame Modul beinhaltet das `<iframe>`-Element, das zur Erstellung von Inlineframes benötigt wird.

Element	Rolle	Typ
`<iframe> ... </iframe>`	Definiert einen Inlineframe	Inlinecontainer

Tabelle 12.33 Elemente des iFrame Moduls

12.5.20 Das Intrinsic Events Modul

Das Intrinsic Events Modul (*intrinsic* = spezifisch) enthält keine Elemente, sondern fügt einer Reihe von Elementen anderer Module[16] explizit bestimmte, ihrem Verhalten immanente Eventhandler hinzu.

Gleichzeitig wird die **Events Sammlung** des Common-Attributsatzes aktiviert, so dass für alle Elemente, denen der Common-Attributsatz zugeordnet ist (dazu gehören abgesehen von `<frameset>` auch alle Elemente der untenstehenden Tabelle!), folgende Eventhandler-Attribute gültig sind:

```
onclick, ondblclick, onmousedown, onmouseup, onmouseover,
onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
```

Zusätzlich zu diesen Eventhandlern bekommen einige Elemente noch weitere, spezifische Eventhandler hinzugefügt:

16 Die Wirkung des Moduls hängt daher von der Gesamtkombination der gewählten Module ab.

Attribut	Rolle	erweitert den Attributsatz für
onblur	Wegnahme des Fokus von einem Element (Link oder Eingabefeld)	<a> <area /> <button> <label> <input /> <select> <textarea>
onfocus	Der Fokus wird auf ein Element (Link oder Eingabefeld) gerichtet	<a> <area /> <button> <label> <input /> <select> <textarea>
onload	Nach beendetem Laden des Dokuments	<body> <frameset>
onunload	Beim Entladen des Dokuments aus dem Fenster oder Frame	<body> <frameset>
onreset	Beim Zurücksetzen der Eingabefelder eines Formulars auf die Vorgabewerte	<form>
onsubmit	Beim Abschicken der Werte eines Formulars	<form>
onchange	Nach erfolgter Änderung des Inhalts eines Eingabefeldes	<input /> <select> <textarea>
onselect	Beim Auswählen des Inhalts eines Eingabefeldes	<input /> <textarea>

Tabelle 12.34 Attributadditionen durch das Intrinsic Events Modul

Anmerkung: Soll das Intrinsic Events Modul auf Formular-Elemente wirken, so muss entweder das Basic Forms Modul oder das Forms Modul eingebunden sein. Analoges gilt für das Frames Modul (`<frameset onload="..." >` oder das Client-Side Image Map Modul (`<area onblur="..." />`).

12.5.21 Das Legacy Modul

Das Legacy Modul (*legacy* = Erbe) enthält all jene Elemente bzw. Attribute, deren Verwendung unerwünscht ist, da sie gegen das Prinzip der strikten Trennung von Inhalten und deren Präsentation verstoßen (dass dies in anderen Modulen zum Teil aufgeweicht wird, steht auf einem anderen Blatt).

Einerseits werden diese Definitionen benötigt, um überhaupt eine vollständige Nachbildung von HTML 4.01 in der *transitional*-Ausprägung erstellen zu kön-

nen. Andererseits sollen auch Dokumente, die diese Elemente und Attribute verwenden, validierbar sein, solange sie ansonsten regelkonform zu XML sind.

Folgende neun Elementdeklarationen sind Teil des Legacy Moduls:

Element	Rolle	Typ
`<basefont />`	Definiert Basisschrifttyp, -farbe und -größe für den folgenden Dokumentbereich	Leeres Element; Inlinekontext

`<center>`
`...`
`</center>`
	Zentriert den eingeschlossenen Containerinhalt	Blockcontainer

`<dir>`
`...`
`</dir>`
	Definiert eine Directory List (Verzeichnisliste)	Blockcontainer

`<font> ...` `</font>`	Präsentiert den eingeschlossenen Textinhalt durch Angabe von Schriftgröße, -typ und -farbe	Inlinecontainer
`<isindex />`	Erzeugt ein einzeiliges Inputfeld	Leeres Element

`<menu>`
`...`
`</menu>`
	Definiert eine Navigationsliste (*menu list*)	Blockcontainer

`<s> ... </s>`	"strikethrough"; präsentiert den eingeschlossenen Inhalt durchgestrichen	Inlinecontainer
`<strike> ...` `</strike>`	"strikethrough"; präsentiert den eingeschlossenen Inhalt durchgestrichen (Alias zu `<s>`)	Inlinecontainer
`<u> ... </u>`	"underline"; präsentiert den eingeschlossenen Textinhalt mit Unterstrich	Inlinecontainer

Tabelle 12.35 Elemente des Legacy Moduls

Folgende Attributdeklarationen, die gegebenenfalls die Attributsätze von Elementen anderer Module erweitern, sind im Legacy Modul enthalten:

Attribut	Rolle	erweitert den Attributsatz für

`align`	Horizontale Ausrichtung des Inhalts des Elements gegenüber dem Container; für `<img />` und `<table>` mit den Werten `"left"` oder `"right"` als Float-Attribut; bezeichnet für `<table>` mit Wert `"center"` die Zentrierung innerhalb der Seite	`<caption>`
		`<div>`
		`<h1> ... <h6>`
		`<hr />`
		`<img />`
		`<input>`
		`<legend>`
		`<p>`
		`<table>`

Tabelle 12.36 Attributadditionen durch das Legacy Modul

Attribut	Rolle	erweitert den Attributsatz für
alink	Die Farbe eines aktiven Hyperlinks Wert: RGB-Farbangabe in Hex-Schreibweise oder Farbbezeichner	<body>
background	Quelle des Hintergrundbilds des Dokuments	<body>
bgcolor	Hintergrundfarbe; Wert: RGB-Farbangabe in Hex-Schreibweise oder Farbbezeichner	<body> <table> <td> <th> <tr>
border	Umrandung des Bildes; Breite in Pixel angegeben	<img />
clear	Umbruch nach Float-Elementen; Werte: "left", "right", "all"	
compact	"kompakte" Darstellung eines Listenelements, indem die oberen und unteren Margins verringert werden	<dl>
height	Die Höhe einer Tabellenzelle in Pixel oder Prozent	<td> <th>
hspace	Horizontaler Freiraum auf beiden Seiten eines Images	<img />
language	Deklariert den Sprachdialekt eines Scriptblockes	<script>
link	Die Farbe eines unbesuchten Hyperlinks; Wert: RGB-Farbangabe in Hex-Schreibweise oder Farbbezeichner	<body>
noshade	Deaktiviert dreidimensionale Darstellung einer horizontalen Trennlinie	<hr />
nowrap	Deaktiviert den Zeilenumbruch in einer Tabellenzelle	<td> <th>
size	Die Dicke (Höhe) einer horizontalen Trennlinie	<hr />
start	Definiert den Startwert der Nummerierung einer geordneten Liste	
text	Schriftfarbe des Dokuments; Wert: RGB-Farbangabe in Hex-Schreibweise oder Farbbezeichner	<body>
type	Definiert den Nummerierungstyp bzw. das Aufzählungszeichen der Listenelemente geordneter oder ungeordneter Listen	<dl>

Tabelle 12.36 Attributadditionen durch das Legacy Modul (Forts.)

Attribut	Rolle	erweitert den Attributsatz für
value	Forciert bestimmten Nummerierungswert eines Listenelements einer Ordered List	
vlink	Die Farbe eines besuchten Hyperlinks; Wert: RGB-Farbangabe in Hex-Schreibweise oder Farbbezeichner	<body>
vspace	Vertikaler Freiraum über und unter einem Image	<img />

width	Breite eines Elements in Pixel oder Prozent in Bezug auf umliegenden Containers (bei <pre> maximale Zeichenzahl pro Zeile)	<hr />
		<pre>
		<td>
		<th>

Tabelle 12.36 Attributadditionen durch das Legacy Modul (Forts.)

12.5.22 Das Link Modul

Das **Link Modul** enthält den <link />-Tag, der zur Verknüpfung externer Ressourcen mit dem XHTML-Dokument dient.

Element	Rolle	Typ
<link />	Bindet eine externe Ressource (Stylesheet o.ä.) ein	Leeres Element; Headkontext

Tabelle 12.37 Elemente des Link Moduls

12.5.23 Das Meta Information Modul

Das **Meta Information Modul** enthält den <meta />-Tag zur Hinzufügung von Metainformationen zu einem Dokument.

Element	Rolle	Typ
<meta />	Definiert Metainformationen zum Dokument wie Beschreibung, Keywords, Default-Zeichensatz oder Meta-Angaben nach Dublin Core[17]	Leeres Element; Headkontext

Tabelle 12.38 Elemente des Meta Information Moduls

Das <meta>-Element erhält aus dem Common-Attributsatz lediglich die I18N-Collection. Darüber hinaus stehen die Attribute content, scheme, sowie alternativ http-equiv und name zur Verfügung

17 http://dublincore.org/

12.5.24 Das Name Identification Modul

Dieses Modul fügt einigen Elementen das (missbilligte) `name`-Attribut hinzu, das ansonsten zugunsten des `id`-Identifiers fallen gelassen wurde, der ebenso oder besser geeignet ist, als Fragment Identifier bzw. unverwechselbarer Identifier eines Elements zu dienen. Der Einsatz dieses Moduls wird nicht empfohlen.

Attribut	Rolle	erweitert den Attributsatz für

`name`	Namensidentifier; verglichen mit dem `id`-Identifier (vom Typ `ID`) weniger reglementiert	`<a>`
		`<applet>`
		`<form>`
		`<frame />`
		`<iframe>`
		`<img />`
		`<map>`

Tabelle 12.39 Attributadditionen durch das Name Identification Modul

Anmerkung: Auch *ohne* Einbinden des Name Identification Moduls ist das `name`-Attribut für Formulareingabeelemente (`<input />`, `<textarea>`, `<select>`) deklariert (*nicht* jedoch für das `<form>`-Element selbst).

12.5.25 Das Object Modul

Das Object Modul enthält den `<object>`-Container, der zur Einbettung verschiedenartiger Objekte in die XHTML-Seite dient.

Element	Rolle	Typ
`<object> ...` `</object>`	Fügt ein Objekt in die Seite ein	Inlinecontainer
`<param />`	Übergibt Parameter an den umgebenden `<object>`-Container	Leeres Element

Tabelle 12.40 Elemente des Object Moduls

Anmerkung: Das Object Modul ist quasi funktionsgleich zum (missbilligten) Applet Modul, das alternativ den `<applet>`-Container deklariert, der jedoch auf das Einbinden von Java-Applets beschränkt ist.

12.5.26 Das Scripting Modul

Das Scripting Modul stellt zwei Elemente zur Verfügung, die zum Einfügen von Scripten in eine XHTML-Seite dienen.

Element	Rolle	Typ
`<script> ... </script>`	Definiert einen Scriptcontainer	Inlinecontainer
`<noscript> ... </noscript>`	Definiert einen alternativen Inhalt für den Fall, dass ein Script nicht interpretiert wird	Blockcontainer

Tabelle 12.41 Elemente des Scripting Moduls

Das `<noscript>`-Element erhält den Common-Attributsatz. Für `<script>` werden dagegen lediglich elementspezifische Attribute deklariert: `charset`, `src`, `type` und `defer` (mit Wert `"defer"`) sowie `xml:space` (mit Wert `"preserve"`).

Anmerkung: Das (missbilligte) Attribut `language` für `<script>` ist hier nicht enthalten, es ist jedoch Teil des Legacy Moduls.

12.5.27 Das Server-Side Image Map Modul

Das Modul fügt einem Bild oder einem als Submitbutton eines Formulars dienendem Bild das `ismap`-Attribut hinzu.

Attribut	Rolle	erweitert den Attributsatz für
`ismap`	Definiert das Element als serverseitig ausgewertete Imagemap, deren Click-Koordinaten ausgewertet werden	`<img />` `<input type="image"/>`

Tabelle 12.42 Attributadditionen durch das Server-Side Image Map Modul

12.5.28 Das Style Attribut Modul

Das Style Attribut Modul fügt allen Elementen, denen der Common-Attributsatz zugeordnet ist, ein `style`-Attribut hinzu. Dies betrifft die Mehrzahl der XHTML-Elemente.

Ausnahmen und Inkonsistenzen bei der Zuweisung des style-Attributs

Nicht zugewiesen wird das `style`-Attribut den Strukturtags `<html>` und `<head>`, Headcontent-Elementen wie `<title>`, `<meta />`, `<base />`, `<basefont />`, `<style>` und `<script>`, den iFrame- und Frameset-Elementen `<iframe>`, `<frame />` und `<frameset>` sowie weiteren Elementen wie `<map>`, `<applet>` und `<param>`, `<font>`, `<isindex />`, `<br />` und `<bdo>`.

Ein paar **Inkonsistenzen** der Moduldeklaration sind gegenüber den Spezifikationen HTML 4.01 und XHTML 1.0 zu bemerken: So besitzt der `<font>`-Tag dort ein `style`-Attribut, ebenso der `<frameset>` und `<frame>`-Tag.

Attribut	Rolle	erweitert den Attributsatz für
style	Ermöglicht eine Inlinestyledeklaration für einen Elementcontainer	alle Elemente mit Common Attributsammlung

Tabelle 12.43 Attributadditionen durch das Style Attribute Modul

Anmerkung: Das style-Attribut unterläuft den Sinn von Stylesheets, die Präsentationsdeklarationen vom Dokument zu lösen, und sollte deshalb nicht verwendet werden.

12.5.29 Das Style Sheet Modul

Das Style Sheet Module enthält das <style>-Element, das zur Deklaration von dokumentinternen CSS-Deklarationen oder zur Verknüpfung von Stylesheets verwendet wird.

Element	Rolle	Typ
<style> ... </style>	Enthält CSS-Deklarationen oder die Importanweisung eines externen Stylesheets (@import)	Unsichtbarer Container (Dokumentkopf)

Tabelle 12.44 Elemente des Style Sheet Moduls

12.5.30 Das Target Modul

Das Target Modul definiert, welchen Elementen ein target-Attribut hinzugefügt wird. Welche Elemente betroffen sind, ist abhängig von den ansonsten eingebundenen Modulen. Mindestens bezieht sich das Modul auf den <a>-Tag, da das Hypertext Modul in jedem zur XHTML-Familie gehörenden Sprachkonstrukt vorhanden ist.

Attribut	Rolle	erweitert den Attributsatz für
target	Deklariert das Zielfenster eines Hyperlinks, i.d.R einen Frame; öffnet außerhalb des Framekontexts ein neues Browserfenster	**<a>** <area /> <base /> <form></form> <link />

Tabelle 12.45 Attributadditionen durch das Target Modul

Anmerkung: Damit das Target Modul über den <a>-Tag hinaus Wirkung zeigt, müssen diejenigen Zusatzmodule eingebunden sein, die dessen weitere Zieltags definieren:

- für das `<area />`-Element das **Client-side Image Map Modul**
- für das `<base />`-Element das **Legacy Modul**
- für das `<form>`-Element das **Basic Forms** oder das **Forms Modul**
- für das `<link />`-Element das **Link Modul**

Target auch ohne Frames Modul?

Gewöhnlich wird das Target Modul in Zusammenhang mit einem ebenfalls eingebundenen **Frames Modul** eingesetzt. Es wird jedoch in der Spezifikation ausdrücklich darauf hingewiesen, dass dies nicht *zwangsläufig* so sein muss – der Einsatz von Target auch in Abwesenheit von Framesets (zum Öffnen neuer Browserfenster) ist hiermit gestattet.

12.6 Die Hostlanguage: XHTML Basic

Das Vokabular von **XHTML Basic** basiert auf den Modulen, die in der Spezifikation zur Modularisierung von XHTML festgelegt sind.

Die W3C-Spezifikation zu XHTML Basic existiert in der aktuellen Version seit Dezember 2000, wurde also zu einem Zeitpunkt verabschiedet, als die Spezifikation zur **Modularisierung von XHTML** noch als »Candidate Recommendation« galt. Letztere erwies sich jedoch als stabil genug, um an XHTML Basic keine weiteren Anpassungen vornehmen zu müssen.

Enthalten sind

- die vier Core-Module:

  Structure Module, Text Module, Hypertext Module, List Module
- die zwei Basic Module für **Formulare** und **Tabellen**:

  Basic Forms Module, Basic Tables Module
- weitere Module für Bilder, Objekte, Metainformationen:

  Image Module, Object Module, Metainformation Module, Link Module, Base Module

Durch die Einbeziehung der vier Core-Module zählt XHTML Basic zu den **XHTML Host-Sprachen**. Der gegenüber XHTML merklich reduzierte Umfang macht die Sprache unmittelbar geeignet für mobile Endgeräte, die nur einen kleinen Sprachumfang unterstützen – namentlich Mobiltelephone, Pager und PDAs, aber auch Set-Top Boxen (WebTV Interfaces).

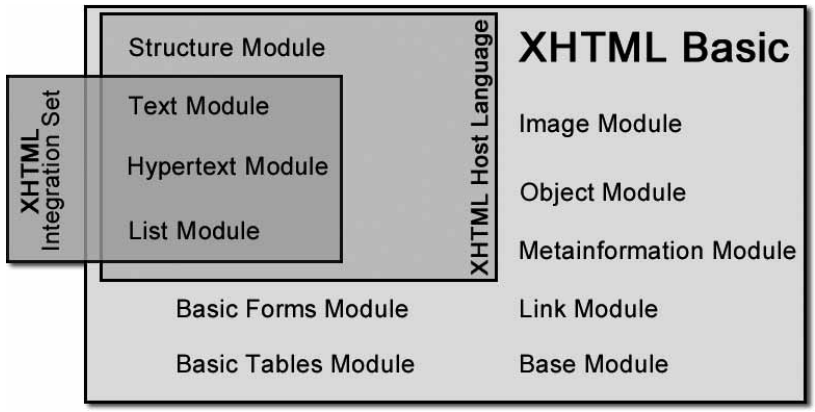

Abbildung 12.7 XHTML Basic umfasst die Host Language Module.

XHTML Basic umfasst folgenden, aus 52 Elementen bestehenden Tagsatz:

`<a> <abbr> <acronym> <address>`
`<base/> <blockquote> <body>  `
`<caption> <cite> <code>`
`<dd> <dfn> <div> <dl> <dt>`
`<em> <form>`
`<h1> <h2> <h3> <h4> <h5> <h6> <head> <html>`
`<img/> <input/>`
`<kbd> <label> <li> <link>`
`<meta/>`
`<object> <ol> <option>`
`<p> <param> <pre> <q>`
`<samp> <select> <span> <strong>`
`<table> <td> <textarea> <th> <title> <tr>`
`<ul> <var>`

Tabelle 12.46 Elemente von XHTML Basic

12.7 Die Zukunft: XHTML 2.0

Bei **XHTML 2.0** handelt es sich nicht um eine direkte Weiterentwicklung von XHTML 1.1, sondern quasi um eine Neu- bzw. Parallelentwicklung, die jedoch den Vorgängerstandard langfristig ablösen soll. Strikte Abwärtskompatibilität ist hier also kein Thema.

Hintergrund der Überlegungen zu diesem neuen Standard ist die Verbesserung der Abbildung hierarchischer Dokumentstrukturen, die mit dem bisherigen XHTML-Vokabular nur ungenügend dargestellt werden kann. Dies spiegelt sich im neu eingeführten `<section>`-Container wider, der Dokumentabschnitte markiert und durch Verschachtelung deren hierarchische Staffelung darstellen kann, sowie den entsprechenden sektionsbezogenen, neutralen Überschriftscontainer `<h>`, der seine Hierarchiestufe aus dem umliegenden `<section>`-Container bezieht. Im gleichen Zuge wurden auch Inhaltsmodelle bestimmter Elemente geändert, so dass beispielsweise der `<p>`-Container auch Tabellen oder Formulare als Teil eines Absatzes enthalten darf.

Die Gruppe der HTML-Formularelemente entfällt und wird durch Elemente aus dem **XForms**-Vokabular ersetzt. Aus XHTML 1.1 übernommen wird die Hinzunahme des **Ruby**-Tagsatzes. Neu sind Elemente, die der Eventerkennung und -behandlung dienen (`<ev:listener>`, `<handler>`). Dies wurde aus der Spezifikation zu **XML Events** abgeleitet. Darüber hinaus gibt es weitere Neuerungen wie den Zeilencontainer `<l>`, Zusammenfassungen für Tabelleninhalte `<summary>` und mehr.

12.7.1 Übersicht über die Elemente von XHTML 2.0

In der geplanten Spezifikation zu **XHTML 2.0** ist weiterhin ein Großteil der Elemente aus *XHTML 1.1 strict* enthalten. Des Weiteren werden neue Elemente definiert und in Form externer Module solche aus anderen Vokabularen in die Sprache integriert.

Es gibt in XHTML 2.0 somit insgesamt 94 Elemente, einige darunter aus externen Namensräumen (Neuzugänge gegenüber XHTML 1.1 fett markiert).

Fünf der sechs Ruby-Elemente sind nicht mitgezählt (daher in eckigen Klammern), da sie nur indirekt über das Content-Modell des `<ruby>`-Containers enthalten sind. Die Anzahl und Art der Elemente mag sich bis zur endgültigen Fertigstellung der Spezifikation noch ändern.

<a> <abbr> **<access>** <address>
<blockcode> <blockquote> <body>
<caption> <cite> <code> <col> <colgroup>
<dd> <dfn> **<di>** <dl> <dt>
<h> <h1> <h2> <h3> <h4> <h5> <h6> **<handler>** <head> <html>

Tabelle 12.47 Elemente von XHTML 2.0 (gemäß Stand der Working Draft vom 27.5.2005)

<img> <kbd> **<l>** <label>   <link>
<meta>
<nl>
<object>
<p> <param> <pre> <quote>
<samp> **<section> <separator>**[18] **<standby>** <style> <sub> **<summary>** <sup>
<table> <tbody> <td> <tfoot> <th> <thead> <title> <tr>
<var>
XML Events Modul
<ev:listener>[19]
XForms-Modul:
<action> <delete> <dispatch> <input> <insert> <group> <load> <message> <model> <output> <range> <rebuild> <recalculate> <refresh> <repeat> <reset> <revalidate> <secret> <select> <select1> <send> <setfocus> <setindex> <setvalue> <submit> <switch> <textarea> <trigger> <upload>
Ruby Modul
<ruby> [<rb>, <rt>, <rp>, <rbc>, <rtc>]

Tabelle 12.47 Elemente von XHTML 2.0 (gemäß Stand der Working Draft vom 27.5.2005) (Forts.)

Frameset-Funktionalität ist in XHTML 2.0 nicht enthalten und soll auch nicht über Module nachgerüstet werden. Stattdessen wird empfohlen, auf **XFrames** zurückzugreifen (diese Spezifikation liegt seit 2002 als Working Draft vor).

12.7.2 Modularisierung von XHTML 2.0

Wie XHTML 1.x ist auch XHTML 2.0 in Sprachmodule unterteilt. Man unterscheidet in XHTML 2.0 vierzehn Module, darunter drei externe.

Core-Module

Die ersten fünf Module können, obwohl sie in der Spezifikation nicht ausdrücklich als solche bezeichnet werden, mit den vier **Core-Modulen** von XHTML 1.x gleichgesetzt werden:

- **XHTML Document Module**
  Es enthält die vier grundlegenden Strukturelemente eines XHTML-Dokuments: <html>, <head>, <title>, <body>.

18 In der Diskussion ist es, den Bezeichner dieses Elements zu <sep> zu vereinfachen.

19 Namensraum: xmlns:ev="http://www.w3.org/2001/xml-events"

- **XHTML Structural Module**
  Dieses Modul enthält die Block-Elemente, die im Dokumentkörper auftreten können: `<address>`, `<blockcode>`, `<blockquote>`, `<div>`, `<h>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<p>`, `<pre>`, `<section>`, `<separator>`.
- **XHTML Text Module**
  Dieses Modul enthält die Inlineelemente, die im Dokumentkörper auftreten: `<abbr>`, `<cite>`, `<code>`, `<dfn>`, `<em>`, `<kbd>`, `<l>`, `<quote>`, `<samp>`, `<span>`, `<strong>`, `<sub>`, `<sup>`, `<var>`.
- **XHTML Hypertext Module**
  Dieses Modul enthält das `<a>`-Element.
- **XHTML List Module**
  Hier enthalten sind die Listenelemente `<dl>`, `<di>`, `<dt>`, `<dd>`, `<label>`, `<nl>`, `<ol>`, `<ul>`, `<li>`.

Diese fünf grundlegenden Module enthalten insgesamt 37 Elemente (die Core-Module von XHTML 1.x umfassen 35 Elemente). Zu beachten ist, dass das jetzige *Document Module* dem *Structure Module* von XHTML 1.x entspricht und das neue *Structural Module* wie das *Text Module* aus der (durchaus sinnvollen) Aufteilung des ehemaligen *Text Modules* hervorgehen. Unverändert ist das *Hypertext Module*. Dem *List Module* wurden neue Elemente hinzugefügt.

Weitere XHTML-Module

- **XHTML Handler Module**
  enthält das `<handler>`-Element, das das in XHTML 2.0 abgeschaffte `<script>`-Element ersetzt.
- **XHTML Image Module**
  enthält das `<img>`-Element.
- **XHTML Metainformation Module**
  enthält das `<meta>` und das `<link>`-Element.
- **XHTML Object Module**
  enthält `<object>`, `<param>` und `<standby>`-Element.
- **XHTML Role Access Module**
  enthält das `<access>`-Element.
- **XHTML Style Sheet Module**
  enthält das `<style>`-Element.

- **XHTML Tables Module**
  enthält die Tabellenelemente `<table>`, `<caption>`, `<summary>`, `<col>`, `<colgroup>`, `<thead>`, `<tfoot>`, `<tbody>`, `<tr>`, `<td>`, `<th>`.

Anmerkung: In XHTML 2.0 ist (derzeit) kein Basic Tables Modul vorgesehen. Formularelemente werden aus dem XForm-Vokabular bezogen, aus dem aber ebenfalls kein analoges Subset definiert ist.

Extern definierte Module

- **Ruby Module**
  enthält das `<ruby>`-Element und damit indirekt dessen potentielle Kindelemente.
- **XForms Module**
  enthält, eingeteilt in verschiedene Untermodule, die Elemente `<model>` (in XForms Core), `<input>`, `<secret>`, `<textarea>`, `<output>`, `<upload>`, `<range>`, `<trigger>`, `<submit>`, `<select>`, `<select1>` (in XForms Form Controls), `<action>`, `<dispatch>`, `<rebuild>`, `<recalculate>`, `<revalidate>`, `<refresh>`, `<setfocus>`, `<load>`, `<setvalue>`, `<send>`, `<reset>`, `<message>`, `<insert>`, `<delete>`, `<setindex>` (in XForms Actions), `<group>` (in XForms Group), `<switch>` (in XForms Switch), `<repeat>` (in XForms Repeat).
- **XML Events Module**
  enthält das Element `<ev:listener>` (im XML Events-Namensraum).

12.7.3 Allgemeiner Attributsatz der XHTML 2.0-Elemente

Vergleichbar mit der Praxis bei der Modularisierung von XHTML 1.0 existiert in XHTML 2.0 ein allgemeiner Attributsatz, der hier ebenfalls als *Common Attribute Collection* bezeichnet wird. Im Gegensatz zur gleichnamigen Attributsammlung in XHTML 1.0 umfasst die Sammlung von XHTML 2.0 jedoch eine weitaus größere Anzahl von Unterabteilungen, die jetzt auch ausdrücklich als »Module« und nicht mehr als »Sammlungen« (*collections*) bezeichnet werden.

Man unterscheidet in XHTML 2.0 dreizehn Attribut-Module:

- **Core-Attribut Modul**
  enthält als Core-Attribute `id`, `layout`, `title` und `class`, wobei das in XHTML 2.0 hinzugefügte Attribut `layout` das Whitespace-Verhalten eines Elements beschreibt (vergleichbar `<pre>`-Containers).
- **I18N-Attribut Modul**
  enthält das Sprachkennungs-Attribut `xml:lang` (im XML-Namensraum).

- **Bi-directional-Attribut Modul**
  enthält das `dir`-Attribut zur Kennzeichnung der Schreibrichtung eines Elementinhalts.
- **Edit-Attribut Modul**
  enthält die Attribute `edit` und `datetime`, um Art und Zeitpunkt der Bearbeitung eines Elementinhalts zu kennzeichnen.
- **Embedding-Attribut Modul**
  enthält die drei Attribute `encoding`, `src` und `srctype`, um Typ und Quelle einer in ein Element eingebetteten externen Ressource zu beschreiben
- **Events-Attribut Modul**
  enthält die sieben Attribute `defaultAction`, `event`, `handler`, `observer`, `phase`, `propagate` und `target`, die im Rahmen eines externen Moduls aus der **XML Events**-Spezifikation abgeleitet sind und sich daher auch in deren Namensraum befinden: Die Attribute erlauben es, einem Element bestimmte Eventerkennungen bzw. Eventbehandlungen zuzuordnen.
- **XForms-Attribut Modul**
  enthält die Attribute aus **XForms**, die Bindungen eines Elements an ein X-Formular beschreiben: `repeat-model`, `repeat-bind`, `repeat-nodeset`, `repeat-startindex`, `repeat-number`.
- **Hypertext-Attribut Modul**
  enthält neun Attribute, die einem Element den Charakter eines Hyperlinks verleihen und diesen genauer beschreiben: `cite`, `href`, `hreflang`, `hrefmedia`, `hreftype`, `nextfocus`, `prevfocus`, `target`, `xml:base`. Hiermit ist Hyperlinkverhalten nicht mehr allein auf das `<a>`-Element beschränkt, sondern steht jedem Element zur Verfügung.
- **Image Map-Attribut Modul**
  enthält die Attribute `usemap`, `ismap`, `shape` und `coords`, die zum Definieren clientseitiger Imagemaps benötigt werden.
- **Media-Attribut Modul**
  enthält das `media`-Attribut zur Unterscheidung von Medientypen.
- **Metainformation-Attribut Modul**
  enthält die Attribute `about`, `content`, `datatype`, `property`, `rel`, `resource`, `restype` und `rev`, mit denen Metainformationen an Elemente gebunden werden können.
- **Role-Attribut Modul**
  enthält das `role`-Attribut, um die »Rolle« eines Elements zu definieren.
- **Style-Attribut Modul**
  enthält das `style`-Attribut zur Definition von Inlinestyles.

Alles darf nun ein Link sein

Eine hervorhebenswerte Folge dieser umfassenden Zusammenstellung ist die Tatsache, dass durch die Hinzunahme der **Hyperlinkattribute** nunmehr jedes beliebige Element Ursprung eines Hyperlinks sein kann (in HTML/XHTML galt dies nur für den `<a>`-Tag). Im Prinzip ist der `<a>`-Tag hierdurch überflüssig geworden, wird aber aus Kompatibilitätsgründen beibehalten.

[zB] *Beispiel*: Die heute gängige Praxis, Menüs in Webseiten mittels Listen zu erzeugen, wird vereinfacht, da nun Listenelemente selbst als Link dienen können. Der Menücharakter der Liste kann durch das neue `<nl>`-Element (*navigation list*) unterstrichen werden:

```
<nl xml:base="http://www.example.com/buchprojekt/">
   <label>Inhaltsverzeichnis</label>
   <li href="kapitel_1.html">Kapitel 1</li>
   <li href="kapitel_2.html">Kapitel 2</li>
   <li href="kapitel_3.html">Kapitel 3</li>
   ...
</nl>
```

12.8 HTML ohne W3C: HTML 5.0

Abseits vom W3C, das sich vollständig von HTML ab- und XHTML zugewendet hat, gibt es dennoch Bestrebungen, HTML auch als SGML-Anwendung weiterzuentwickeln.

12.8.1 Die WHAT WG

Hinter diesem allgemein gern als **HTML 5.0** betitelten Vorhaben steht die **WHAT Working Group**[20], in der sich namhafte Vertreter verschiedener Computer-, Software- und Browserhersteller zusammenfanden. Mitglieder sind unter anderem David Hyatt (Apple, Entwickler des Safari-Browsers), Håkon Wium Lie und Ian Hickson (Opera, Mitentwickler der CSS-Spezifikation), Brendan Eich (CTO Mozilla Corporation, Entwickler von JavaScript) sowie David Baron (Mozilla Corporation, u.a. W3C Web Standards Project) und viele weitere.

12.8.2 »Web Applications« – auch »HTML 5.0« genannt

Zwar wird die schlagwortartige Benennung »HTML 5.0« des Projekts durch die WG nicht wirklich abgelehnt, die offizielle Bezeichnung der konzipierten HTML-

20 »Web Hypertext Application Technology Working Group«; siehe www.whatwg.org.

Weiterentwicklung lautet hingegen **Web Applications** (aktuelle Version 1.0) und soll in Zukunft auch andere Parallelentwicklungen wie **Web Forms** (Formularweiterentwicklung) oder **Web Controls** (Weiterentwicklung CSS und Userinterface) einschließen.

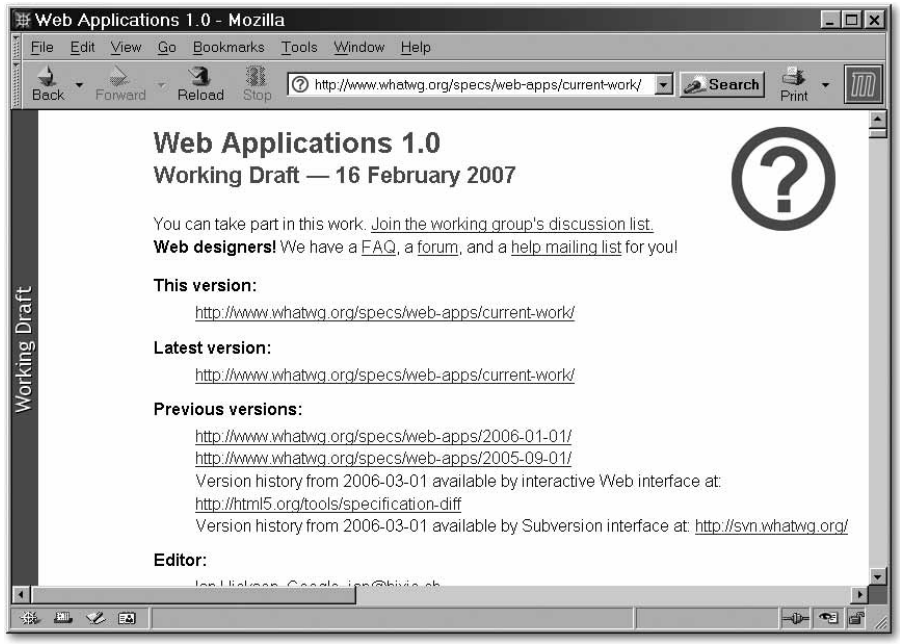

Abbildung 12.8 Der Spezifikationsvorschlag »Web Applications 1.0«

Das Gewicht der Entwicklung liegt primär auf **Abwärtskompatibilität** und stellt sich damit einigen vom W3C vorangetriebenen Standards wie XHTML 2.0 entgegen, die von verschiedenen Seiten als problematisch betrachtetet werden. Der Vorschlag der WHAT WG behält das bestehende HTML-Vokabular (Tags und Attribute) in vollem Umfang bei, verspricht dieses jedoch zum Teil erheblich zu erweitern. Der aktuelle Draft der Spezifikation (Stand 16.2.2007) umfasst etwa 380 Druckseiten, ist also in dieser Hinsicht kein Leichtgewicht.

12.8.3 Geplante Erweiterungen für HTML 5.0

Hierbei wird in vielen Fällen auf Funktionalitäten abgezielt, die in Web 2.0 eine Rolle spielen (wie XMLHttpRequest) oder zum Teil in bestehenden User Agents in proprietärer Form integriert sind. Als wünschenswerte Erweiterungen werden unter anderem genannt:

- Weitergehende Möglichkeiten der Beschreibung von Dokumentstrukturen durch zusätzliche Elemente wie `<header>`, `<section>`, `<article>`, `<byline>`, `<aside>`, `<footer>`
- Elemente zur Navigations- bzw Menügestaltung oder -steuerung: `<nav>`, `<tabbox>`, `<command>`
- Elemente für Zeichenfunktionen und Graphdarstellung (`<canvas>`) oder automatisches Highlighting von Textabschnitten (`<m>`)
- Elemente für Zeitangaben oder Zeitstempel (`<t>`), Kalender (`<calendar>`), Maßeinheiten (`<gauge>`) oder Fortschrittsbalken (`<progress>`)
- Bedingte, alternativ darzustellende Abschnitte in Dokumenten für verbesserte Interface-Gestaltungsmöglichkeiten (`<switch>`)
- Elemente für interaktive Datendarstellung und Databinding (`<datagrid>`, `<datatree>`)
- Definition und Design von Popup- und Kontextmenüs innerhalb des HTML-Dokuments (Bindung eines Elements an eine Kontextmenüdefinition über `IDREF`)
- Erweiterung des Eventhandlings im Hinblick auf Unabhängigkeit von bestimmten Eingabegeräten
- Unterstützung von Drag & Drop sowie Editieren im Browserfenster (Kopieren, Verschieben, Textmanipulation, Clipboardfunktionalität)

Was ist der momentane Stand der Dinge bezüglich »HTML 5.0«?

Der Standard gilt als »Work in Progress«, wird also noch erheblichen Änderungen unterliegen, weshalb er an dieser Stelle nicht näher beschrieben wird. Sie finden den aktuellen Stand der Entwicklung hier: **http://whatwg.org/specs/web-apps/current-work/**.

Einige wenige Vorschläge der WHAT WG sind in aktuellen Browsern versuchsweise bereits implementiert, wie z. B. das `<canvas>`-Element in Safari und Firefox ab 1.5 (dort auch das `ping`- und `role`-Attribut von `<a>`).

[zB] *Beispiel*: In ein `<canvas>`-Element wird per JavaScript ein blaues Dreieck gezeichnet, indem zunächst für ein 2D-Objekt der `canvas` ein Pfad definiert (mit Pixelkoordinaten) und dessen Umriss dann mit der vordefinierten Methode `stroke()` nachgezogen wird. Das Ergebnis sieht in **Firefox 1.5** folgendermaßen aus:

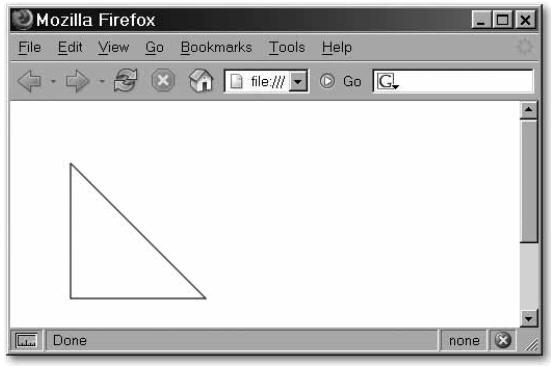

Abbildung 12.9 Das canvas-Element mit gezeichnetem Inhalt in Firefox

```
<html>
<head>
<script type="application/x-javascript">

function zeichnen() {
  var canvas = document.getElementById("canvas");

  var umriss = canvas.getContext("2d");

  umriss.strokeStyle = "blue";

  umriss.beginPath();
  umriss.moveTo(50, 50);
  umriss.lineTo(180, 180);
  umriss.lineTo(50, 180);
  umriss.lineTo(50, 50);
  umriss.stroke();

}
</script>
</head>
<body onload="zeichnen()">
   <!-- ein leeres Canvas-Element: -->
   <canvas id="canvas" width="300" height="300"></canvas>
 </body>
</html>
```

12.9 HTML für mobile Endgeräte

12.9.1 cHTML

Relativ frühzeitig (im Jahr 1997; Revision 1998) wurde dem W3C mit **cHTML** (Compact HTML) eine Abwandlung von HTML (damals in der Version 2.0) für mobile Anwendungen[21] wie Smartphones oder PDAs zur Kenntnis vorgelegt. Es handelt sich daher zwar um keine offiziell vom W3C unterstützte Spezifikation, wenn auch um eine von ihm geduldete. Bei der Sprache handelt es sich um eine recht rücksichtslos abgespeckte Fassung von HTML, die weder Tabellen, noch Stylesheets kennt. Auch ist lediglich die Unterstützung des GIF-Formats für Grafiken vorgesehen, keine Wahl der Font-Darstellung und grundsätzlich nur zwei Farben (Schwarz und Weiß).

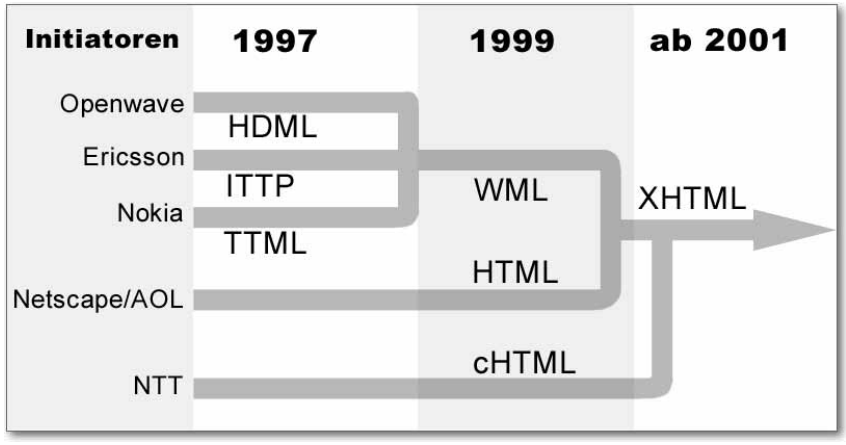

Abbildung 12.10 cHTML im Kontext »HTML für Mobilgeräte«

Der Elementliste von cHTML merkt man deutlich die frühzeitige Ableitung von HTML 2.0 an. Insgesamt sind 34 Elemente enthalten:

```
<A>
<BASE> <BLOCKQUOTE> <BODY> <BR>
<CENTER>
```

Tabelle 12.48 Elemente von cHTML (in eckigen Klammern: entfallen)

21 Etwa zeitgleich propagiert wurde mit HDML (Handheld Device Markup Language) auch ein Sprachvorschlag, der nicht auf HTML basierte.

<DD> <DIR> <DIV> <DL> <DT>
<FORM>
<HEAD> <H1> <H2> <H3> <H4> <H5> <H6> <HR> <HTML>
<IMG> <INPUT> *[<ISINDEX>]*
<LI>
<MENU> <META>
<OL> <OPTION>
<P> *[<PLAINTEXT>]* <PRE>
<SELECT>
<TEXTAREA> <TITLE>
<UL>

Tabelle 12.48 Elemente von cHTML (in eckigen Klammern: entfallen) (Forts.)

Wieviele Elemente hat cHTML wirklich?

Vielleicht auch 36: Die **DTD** von cHTML erlaubt <ISINDEX> als optionalen Inhalt von <HEAD>, definiert dieses Element jedoch nicht. Es dürfte sich bei der Erwähnung um ein Artefakt eines, bei der '98-er Revision entfallenen Elements handeln. Analoges gilt für <PLAINTEXT>, das zwar in der Tagliste enthalten ist, doch in der cHTML-DTD nicht deklariert wird. Dieses Element ist jedoch in iMode HTML 1.0 und Doti 1.0 enthalten.

Nutzen aus cHTML zog in erster Linie der Hauptinitiator, das japanische Mobilfunkunternehmen NTT Mobile Communications Network[22] (später DoCoMo). DoCoMo entwickelte auf der Grundlage von cHTML für seinen, Ende der 90er Jahre eingeführten iMode-Service einige Dialekte, bekannt als **iMode HTML** (Versionen 1.0 bis 4.0).

Spätere Versionen von iMode HTML basieren nicht mehr auf cHTML sondern auf XHTML (iMode XHTML 1.0 bzw. iMode XHTML 1.1). Seit 2002 werden (hier durch E-Plus) iMode-Services auch in Deutschland angeboten; ebenso existieren lokale Anbieter in Holland, Frankreich und Spanien.

Die ebenfalls japanische Konkurrenzfirma TTN leitete 1998 unter der Bezeichnung **Doti 1.0** von cHTML eine eigene, zu iMode HTML 2.0 fast identische Spielart ab. Von cHTML selbst hingegen gibt es, sieht man von der Revision von 1998 ab, keine offiziellen Varianten.

22 NTT: »Nippon Telegraph and Telephone«, in namentlicher Anlehnung an AT&T, zu der jedoch keine Verbindung besteht.

12.9.2 iMode 1.0

NTT DoCoMo begann frühzeitig, die für die Websparte seines iMode-Dienstes verwendete Sprache (also cHTML) als **iMode HTML** zu bezeichnen. Die erste Version von iMode HTML ist im Prinzip nichts anderes als »umgelabeltes« cHTML:

<A>
<BASE> <BLOCKQUOTE> <BODY>
<CENTER>
<DD> <DIR> <DIV> <DL> <DT>
<FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<IMG> <INPUT>
<LI>
<MENU>
<OL> <OPTION>
<P> <PLAINTEXT> <PRE>
<SELECT>
<TEXTAREA> <TITLE>
<UL>

Tabelle 12.49 Elemente von iMode 1.0

Der Tagsatz von iMode 1.0 (34 Elemente) entspricht einer frühen Version von cHTML, was am Vorhandensein von <PLAINTEXT> zu erkennen ist. Dieses Element wird auch in den folgenden drei Versionen beibehalten (bereits seit HTML 2.0 gilt es als »deprecated« – in HTML 4.0 wurde es abgeschafft).

12.9.3 iMode 2.0

Das »Manko« von cHTML, keine Fontbeeinflussung zu bieten, wird in der Folgeversion 2.0 von iMode HTML behoben.

Der Tagsatz wird durch drei Präsentationselemente erweitert, wobei an erster Stelle der <FONT>-Container zu nennen ist, interessanterweise aber auch sowohl <BLINK> als auch <MARQUEE>, bei denen es sich um proprietäre, von Netscape bzw. Microsoft in HTML eingeführte Elemente handelt.

iMode HTML 2.0 besitzt 38 Elemente:

<A>
<BASE> **<BLINK>** <BLOCKQUOTE> <BODY>
<CENTER>
<DD> <DIR> <DIV> <DL> <DT>
<FONT> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<IMG> <INPUT>
<LI>
<MARQUEE> <MENU> **<META>**
<OL> <OPTION>
<P> <PLAINTEXT> <PRE>
<SELECT>
<TEXTAREA> <TITLE>
<UL>

Tabelle 12.50 Elemente von iMode 2.0

12.9.4 iMode 3.0

Der Tagsatz von **iMode 3.0** wird gegen iMode 2.0 durch das <OBJECT>-Element erweitert. Er enthält nunmehr 39 Elemente:

<A>
<BASE> <BLINK> <BLOCKQUOTE> <BODY>
<CENTER>
<DD> <DIR> <DIV> <DL> <DT>
<FONT> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<IMG> <INPUT>
<LI>
<MARQUEE> <MENU> <META>
<OBJECT> <OL> <OPTION>
<P> <PLAINTEXT> <PRE>
<SELECT>
<TEXTAREA> <TITLE>
<UL>

Tabelle 12.51 Elemente von iMode 3.0

12.9.5 iMode 4.0

Gegenüber iMode 3.0 kommt in **iMode 4.0** als möglicher Inhalt des <OBJECT>-Elements der <PARAM>-Tag hinzu.

<A>
<BASE> <BLINK> <BLOCKQUOTE> <BODY>
<CENTER>
<DD> <DIR> <DIV> <DL> <DT>
<FONT> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<IMG> <INPUT>
<LI>
<MARQUEE> <MENU> <META>
<OBJECT> <OL> <OPTION>
<P> **<PARAM>** <PLAINTEXT> <PRE>
<SELECT>
<TEXTAREA> <TITLE>
<UL>

Tabelle 12.52 Elemente von iMode 4.0

12.9.6 Doti 1.0

Das Unternehmen TTN definierte abgeleitet von cHTML seinen eigenen, proprietären Sprachstandard **Doti 1.0** (keine weiteren Versionen), der weitestgehend iMode HTML 2.0 entspricht, wobei noch ein zusätzliches <EMBED>-Element und die beiden Präsentationstags <B> und <I> hinzugefügt wurden.

Doti 1.0 umfasst also insgesamt 41 Elemente:

<A>
<B> <BASE> <BLINK> <BLOCKQUOTE> <BODY>
<CENTER>
<DD> <DIR> <DIV> <DL> <DT>
<EMBED> <FONT> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<I> <IMG> <INPUT>
<LI>
<MARQUEE> <MENU> <META>
<OL> <OPTION>

Tabelle 12.53 Elemente von Doti 1.0

<P> <PLAINTEXT> <PRE>
<SELECT>
<TEXTAREA> <TITLE>
<UL>

Tabelle 12.53 Elemente von Doti 1.0 (Forts.)

12.9.7 HTML 4.0 mobile

Das **HTML 4.0 mobile** Vokabular (55 Elemente) entspricht einem Subset von HTML 4.0 strict.

<A> <ABBR> <ACRONYM> <ADDRESS> <AREA>
<BDO> <BLOCKQUOTE> <BODY>   <BUTTON>
<CITE> <CODE>
<DD> <DFN> <DIV> <DL> <DT>
<EM> <FIELDSET> <FORM>
<H1> <H2> <H3> <H4> <H5> <H6> <HEAD> <HR> <HTML>
<IMG> <INPUT>
<KBD> <LABEL> <LEGEND> <LI> <LINK>
<MAP> <META>
<OBJECT> <OL> <OPTGROUP> <OPTION>
<P> <PARAM> <PRE> <Q>
<SAMP> <SELECT> <SPAN> <STRONG> <STYLE>
<TEXTAREA> <TITLE> <UL> <VAR>

Tabelle 12.54 Elemente von HTML 4.0 mobile

Gegenüber HTML 4.01 strict wird dabei auf die reinen Präsentationselemente sowie auf die Tabellenelemente verzichtet. Darüber hinaus fehlen die Scriptelemente <SCRIPT> und <NOSCRIPT> und das <BASE>-Element (siehe **http://www.w3.org/TR/NOTE-html40-mobile**).

12.10 XHTML für mobile Endgeräte

12.10.1 iMode XHTML 1.0

NTT DoCoMo führte als Nachfolger seiner iMode HTML-Dialekte auch Vokabulare in XHTML ein. **iMode XHTML 1.0** entspricht weitestgehend iMode HTML 4.0, wobei einige veraltete oder proprietäre Elemente wie <BLINK>, <CENTER>, <DIR>, <EMBED>, <FONT>, <MARQUEE>, <MENU> und <PLAINTEXT> entfallen, dafür aber und <style> hinzukommen.

Insgesamt zeigt sich die Sprache standardkonformer:

```
<a>
<base /> <blockquote> <body> <br />
<dd> <div> <dl> <dt>
<form>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<img /> <input />
<li>
<meta />
<object> <ol> <option>
<p> <param /> <pre>
<select> <span> <style>
<textarea> <title>
<ul>
```

Tabelle 12.55 Die Elemente von iMode XHTML 1.0

12.10.2 iMode XHTML 1.1

iMode XHTML 1.1 unterschiedet sich von der Vorgängerversion 1.0 nur im Attributsatz einiger Elemente – so erhält beispielsweise das `<a>`-Element noch weitere proprietäre Attribute, die für NTT-Mobilgeräte verwendet werden.

```
<a>
<base /> <blockquote> <body> <br />
<dd> <div> <dl> <dt>
<form>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<img /> <input />
<li>
<meta />
<object> <ol> <option>
<p> <param /> <pre>
<select> <span> <style>
<textarea> <title>
<ul>
```

Tabelle 12.56 Die Elemente von iMode XHTML 1.1

12.10.3 XHTML mobile profile

Das von der **Open Mobile Alliance** initiierte und als Ablösung von WML (Wireless Markup Language) 1.0 und 1.2 gehandelte »XHTML Mobile Profile« ist zwar kein echter Nachfolger der genannten Sprachen, wird aber dennoch gerne als »WML 2.0« bezeichnet.

`<a> <abbr> <acronym> <address>`
`<b> <base /> <big> <blockquote> <body>  `
`<caption> <cite> <code>`
`<dd> <dfn> <div> <dl> <dt>`
`<em> <fieldset> <form>`
`<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>`
`<i> <img /> <input />`
`<kbd> <label> <li> <link />`
`<meta />`
`<object> <ol> <optgroup> <option>`
`<p> <param /> <pre> <q>`
`<samp> <select> <small> <span> <strong> <style>`
`<table> <td> <textarea> <th> <title> <tr>`
`<ul> <var>`

Tabelle 12.57 Die Elemente von XHTML mobile profile

Das Vokabular (60 Elemente) entspricht im Wesentlichen dem von XHTML Basic, enthält aber zusätzlich mit `<big>`, `<small>`, `<hr/>`, `<b>` und `<i>` noch die meisten Elemente aus dem Presentation Modul sowie `<style>` aus dem Style Modul und `<optgroup>`, `<fieldset>` aus dem Forms Modul.

Anmerkung: Das »offizielle« WML 2.0 (Open Mobile Alliance) besteht hingegen aus XHTML Basic mit einer Reihe hinzugefügter Legacy-Elemente aus HTML einerseits und (aus Gründen der Abwärtskompatibilität) den alten WML-Elementen andererseits, die sich aber im WML-Namensraum befinden.

Dieses Kapitel wirft einen kurzen, einführenden Blick auf XFrames und XForms, zwei zukunftsträchtige XML-Applikationen, die als Ergänzung von XHTML dienen.

13 XForms und XFrames

Mit XForms und XFrames existieren zwei auf XML basierende Spezifikationen, die als Ergänzungen zu XHTML 1.1 bzw. 2.0 dienen. XFrames ist dafür vorgesehen, die in XHTML 1.1 wegfallende Framefunktionalität zu ersetzen. XForms wiederum dient in XHTML 2.0 als Ersatz für das Modul »Forms«.

Während Anwender noch auf eine funktionierende XFrames-Implementierung warten müssen, lassen sich XForms bereits einsetzen: Neben serverseitigen Web Applikationen (die XForms jedoch in HTML-kompatible Formate umwandeln) existieren hierfür in Form des Alternativbrowsers X-Smiles und seit neuestem als Firefox-Erweiterung auch clientseitige Anwendungen.

13.1 XForms

An HTML-Formularen wird bemängelt, dass zum einen weder eine Möglichkeit der Validierung von Eingaben ohne externe Hilfsmittel (JavaScript) noch eine Angabe von Datentypen der Eingabedaten besteht. Zum anderen wird in einem statischen HTML-Formular die Struktur der übertragenen Daten (`name-value`-Pärchen) durch diejenige des Formulars (Zahl und Benennung der Eingabefelder) bereits festgelegt.

Ein grundlegender Unterschied zu HTML-Formularen ist, dass XForms, die als Spezifikation eine XML-Anwendung darstellen, selbst mit Daten in XML-Format arbeiten. Dies betrifft sowohl die von einem XFormular versandten oder lokal gespeicherten Daten als auch die Vorhaltung von Werten für Feldvorbelegungen. Gleichzeitig trennt XForms die Datenlogik von der Darstellung der Benutzerschnittstelle, ist also prinzipiell unabhängig vom Ausgabegerät. Zur Erfassung und Verarbeitung des Eingabevorgangs existiert ein komplexes, auf XML-Events basierendes Eventmodell.

Bei XForms handelt es sich jedoch nicht um eine eigenständige Sprache, sondern nur um ein Modul, das in eine Hostlanguage eingebettet werden muss. Dies wird normalerweise **XHTML** sein, sofern das XFormular direkt an den Client ausgeliefert wird, um dort verarbeitet zu werden.[1]

13.1.1 Anwendungen und Implementierungen von XForms

Die Entwicklung von XForms dauert bereits eine Reihe von Jahren an – der erste Working Draft datiert vom April 2000. Die Entwicklung von XForms-Implementierungen beschleunigte sich nach der Einigung, statt eigener Datentypen diejenigen aus dem damals ebenfalls noch in Entwicklung befindlichen XML Schema einzuführen. Inzwischen existieren eine Reihe von Anwendungen, die über eine reine Testimplementierung hinaus sind.

Clientseitige XForms-Verarbeitung

Es existieren einige Anwendungen, die XForms unmittelbar auf Clientseite verarbeiten können. Bislang wird dabei jedoch in den meisten Fällen nicht der volle Standard unterstützt.

Mozilla XForms Projekt

Das Mozilla XForms Projekt stellt für **Firefox** ab Version 1.5 und **SeaMonkey** 1.0 eine Erweiterung zur Verfügung, die XForms interpretiert (**http://www.mozilla.org/projects/xforms/**).

X-Smiles

Die vollständigste clientseitige Implementierung von XForms existiert derzeit in **X-Smiles**, einem finnischen Browserprojekt, das gleichzeitig eine Testimplementierung von SMIL, XHTML 2.0, SVG und weiteren fortgeschrittenen Technologien ist. Die aktuelle Version X-Smiles 1.0 kann nach Registrierung unter folgender Webadresse kostenlos bezogen werden: **http://www.x-smiles.org/**.

Weiterhin zu erwähnen sind der XForms Browser des dänischen Anbieters **PicoForms** (**www.picoforms.com**), **FormFaces** (**www.formfaces.com**), eine für nichtkommerzielle Anwender freie Software, die XForms in Ajax nachmodelliert, sowie das kostenlose Internet Explorer-Plug-in **FormsPlayer** (**www.formsplayer.com**).

Eine Auflistung weiterer XForms-Implementierungen finden Sie beim W3C (**http://www.w3.org/MarkUp/Forms/#implementations**).

---

1 Die Mozilla-Erweiterung verlangt explizit, dass Webseiten, die XForms enthalten, als XHTML ausgeliefert werden und nicht etwa als HTML.

Serverseitige XForms-Verarbeitung

Die serverseitige Verarbeitung von XForms ermöglicht die Zurverfügungstellung von XForms-Funktionalität auch an Clients, die selbst nicht in der Lage sind, XForms zu interpretieren. Dies geschieht in der Regel durch Umwandlung der XFormulare in herkömmliche HTML-Formulare oder über Java-Applets.

Chiba Project

Einer der bedeutenden Ansätze in dieser Richtung ist das Sourceforge-Projekt Chiba, das in mehreren Versionen vorliegt: **Chiba Core** in Form eines Java-Beans, **Chiba Web** als Servlet mit Ajax-Unterstützung und **Chicoon**, das die Chiba-Funktionalität in Cocoon[2] integriert. Weiterhin existiert eine Clientversion, **Convex**, die als Applet im Internet Explorer 6.0 und 7.0 lauffähig ist.

Sie erreichen das Chiba Project unter **http://chiba.sourceforge.net/**.

Entwicklungsumgebungen für XForms

Eine spezialisierte kommerzielle Entwicklungsumgebung (IDE) für XForms bietet Focus Software in Form des Editors **XFormation** an. Eine auf 30 Tage eingeschränkte Testversion kann unter folgender Adresse bezogen werden: **http://www.xformation.com**.

13.1.2 Die Elemente von XForms

Die XForms-Spezifikation enthält in ihrer »Second Edition« vom März 2006 die große Anzahl von 46 Elementen, die ihrerseits in Module organisiert sind.

`<action>`, `<alert>`, `<bind/>`, `<case>`, `<choices>`, `<copy/>`
`<delete/>`, `<dispatch>`, `<extension>`, `<filename/>`
`<group>`, `<help>`, `<hint>`
`<input>`, `<insert/>`, `<instance>`, `<item>`, `<itemset>`
`<label>`, `<load>`
`<mediatype/>`, `<message>`, `<model>`
`<output>`
`<range>`, `<rebuild>`, `<recalculate>`, `<refresh>`, `<repeat>`, `<reset>`, `<revalidate>`
`<secret>`, `<select>`, `<select1>`, `<send>`, `<setfocus>`, `<setindex/>`, `<setvalue>`, `<submission>`, `<submit>`, `<switch>`

Tabelle 13.1 Elemente von XForms

2 Cocoon: Open-Source Web-Applications Framework von Apache.org. An dieser Stelle erlaube ich mir, Stephan Niedermeiers hervorragendes Buch *Cocoon 2 und Tomcat* (Galileo Press) zu diesem Thema zu empfehlen.

<textarea>, <toggle/>, <trigger>
<upload>, <value>

Tabelle 13.1 Elemente von XForms (Forts.)

13.1.3 Der Namensraum von XForms

Die Elemente von XForms befinden sich grundsätzlich im für sie vorgesehenen Namensraum **http://www.w3.org/2002/xforms**. Jede Anwendung, die XFormulare ausführen soll, muss diesen Namenraum erkennen und anhand davon XForms-Elemente und -Attribute identifizieren können.

Je nach Kontext wird der Namensraum günstigerweise an ein Präfix gebunden – setzt man XForms in einem XHTML-Dokument ein, so ist es in der Regel weniger aufwendig, die XForms-Elemente statt die XHTML-Elemente zu kennzeichnen.

Namensraumpraxis in den Beispielen

In den später folgenden Beispielen werden die Elemente mit dem Präfix `xfm` versehen. Weitere gängige Präfixe sind `xf` und `xform`.

13.1.4 Die Module von XForms

Analog zur Praxis in XHTML sind die Elemente und Teile der Attribute von XForms in Module gegliedert. Dies ermöglicht es, für die Konzeption von Anwendungen oder die Definition eigener Markupsprachen nur Teile des Sprachumfangs zu verwenden. Unabdingbar sind dabei das **XForms Core Modul** und das **XForms Form Controls Modul**.

XForms Core Module

Die Elemente des XForms Core Modul, an erster Stelle das Element `<model>`, definieren die Funktion des Formulars und die Struktur der Daten, die es erfassen soll. Das Modul enthält vier Elemente.

Element	Rolle	Typ
<model> ... </model>	Enthält die Formulardefinition <instance> sowie <submission>, <bind> und Elemente des Action-Moduls	Container
<instance> ... </instance>	Enthält die Definition der Datenstruktur und die Vorbelegungswerte der Felder; Inhaltsmodell ANY	Container

Tabelle 13.2 Elemente des XForms Core Module

Element	Rolle	Typ
`<submission> ... </submission>`	Bestimmt Art des Datenversands; wird von `<submit>`-Element referenziert; darf Elemente des Action-Moduls enthalten	Container
`<bind> ... </bind>`	Bindet ein Feld an ein Datenmodell (Nodeset); darf `<bind>`-Elemente enthalten	Container

Tabelle 13.2 Elemente des XForms Core Module (Forts.)

XForms Form Controls Module

Die Elemente des XForms Form Controls Moduls erzeugen das eigentliche User Interface und enthalten sämtliche Eingabeelemente. Das Modul umfasst 19 Elemente.

Element	Rolle	Typ
`<alert> ... </alert>`	Definiert eine für ein Eingabeelement auszugebende Fehlermeldung; enthält neben Zeichendaten (`PCDATA`) optional `<output>`-Elemente	Container
`<choices> ... </choices>`	Gruppiert Auswahlmöglickeiten innerhalb von `<select>` oder `<select1>`; enthält `<item>`, `<itemset>` oder `<choices>`	Container
`<filename />`	Kindelement von `<upload>`; bindet den Dateinamen eines Uploads an ein Attribut	Leeres Element
`<help> ... </help>`	Erzeugt einen Hilfetext für ein Eingabeelement; enthält neben Text (`PCDATA`) optional `<output>`-Elemente	Container
`<hint> ... </hint>`	Erzeugt einen Tooltip für ein Eingabeelement; enthält neben Zeichendaten (`PCDATA`) optional `<output>`-Elemente	Container
`<input> ... </input>`	Eingabefeld; Präsentation bestimmt durch zusätzliche Attribute oder Datentypbindung; enthält `<label>` und potentiell weitere Elemente	Container
`<item> ... </item>`	Auswahlmöglickeit innerhalb von `<select>` oder `<select1>`; gruppierbar mit `<choices>`; enthält `<label>`, `<value>` und potentiell weitere Elemente	Container
`<label> ... </label>`	Enthält eine Beschreibung für ein Eingabeelement; enthält neben Zeichendaten (`PCDATA`) optional `<output>`-Elemente	Container
`<mediatype />`	Kindelement von `<upload>`; bindet den Mediatyp eines Uploads an ein Attribut	Leeres Element

Tabelle 13.3 Elemente des XForms Form Controls Module

Element	Rolle	Typ
<output> ... </output>	Ausgabe über Bindung referenzierter Inhalte oder Ergebnis von XPath-Ausdruck; enthält maximal <label>	Container
<range> ... </range>	Auswahl aus Wertebereich mit Minimum (@start) und Maximum (@end) in festlegbarer Schrittweite (@step);	Container
<secret> ... </secret>	Eingabefeld in Form eines Passwortfeldes mit Verschleierung der Eingabe	Container
<select> ... </select>	Auswahlfeld mit Mehrfachauswahl; Präsentation bestimmt durch zusätzliche Attribute oder Datentypbindung	Container
<select1> ... </select1>	Auswahlfeld mit Einffachauswahl; Präsentation bestimmt durch zusätzliche Attribute oder Datentypbindung	Container
<submit> ... </submit>	Erzeugt einen Submit-Button; muss an ein <submission>-Element gebunden sein	Container
<textarea> ... </textarea>	Erzeugt einen mehrzeiligen Texteingabebereich, analog zu <html:textarea>	Container
<trigger> ... </trigger>	Entspricht <html:button>; dient als User Interface, um Events auszulösen	Container
<upload> ... </upload>	Erzeugt ein Fileupload-Feld; potentiell mit Auswahl (Quelle, Mediatyp)	Container
<value> ... </value>	Der Wert eines innerhalb einer Auswahl angebotenen <item>	Container

Tabelle 13.3 Elemente des XForms Form Controls Module (Forts.)

XForms Action Module

Das Modul enthält Elemente, mit deren Hilfe alle an Eingabeelemente gebundenen XForms-Events auch separat, beispielsweise abhängig von bestimmten Eingaben, getätigt werden können. So kann ein xforms-submit-Event entweder durch Anwählen eines <submit> Elements ausgelöst werden oder direkt, ohne weitere Benutzeraktion durch ein <send>-Element innerhalb einer Aktionenliste.

Das Modul enthält zwölf Elemente.

Element	Rolle	Typ
<action> ... </action>	Eine Aktionenliste; enthält, nacheinander auszuführend, ein oder mehrere Elemente des Action-Moduls	Container
<dispatch />	Verknüpft (über @target) einen Event mit einem beliebigen Element	Leeres Element

Tabelle 13.4 Elemente des XForms Action Module

Element	Rolle	Typ
`<load />`	Lädt mit `@resource` eine Ressource über einen XLink, der bei der Aktion ausgeführt wird	Leeres Element
`<message> ... </message>`	Enthält Zeichendaten (`PCDATA`), `<label>` oder `<output>`-Elemente	Container
`<rebuild />`	Lädt das XFormular neu und führt erforderliche Rebuilds aus	Leeres Element
`<recalculate />`	Lädt das XFormular neu und führt erforderliche Neuberechnungen (*recalculates*) aus	Leeres Element
`<refresh />`	Lädt das XFormular neu	Leeres Element
`<reset />`	Setzt das XFormular auf die Ausgangswerte zurück	Leeres Element
`<revalidate />`	Lädt das XFormular neu und führt erforderliche Validierungen aus	Leeres Element
`<send />`	Leitet Verschicken der XForms-Daten als Aktion ein; muss an `<submission>`-Element gebunden sein	Leeres Element
`<setfocus />`	Setzt den Fokus auf ein beliebiges Eingabeelement	Leeres Element
`<setvalue> ... </setvalue>`	Setzt einen Datenknoten explizit auf einen literalen Wert (`PCDATA`) oder wertet einen mit `@value` übergebenen XPath-Ausdruck gegenüber der Dateninstanz aus	Container

Tabelle 13.4 Elemente des XForms Action Module (Forts.)

XForms Extension Module

Dieses Modul umfasst nur ein einziges Element. Es bildet für Applikationen einen Transportcontainer für eigenständige Erweiterungen.

Element	Rolle	Typ
`<extension> ... </extension>`	Enthält anwendungsspezifische Erweiterungen des Formulars, die einen eigenen Namensraum benötigen; Inhaltsmodell `ANY`	Container

Tabelle 13.5 Elemente des XForms Extension Module

XForms erweitern

Trotz des großen Funktionsumfangs kann es sinnvoll sein, eine Lösung besonders gelagerter Probleme in XForms durch eine Erweiterung zu erleichtern bzw. diese erst möglich zu machen. Einige Beispiele für denkbare Erweiterungen bietet **eXforms**, das neben Variablen und Sortierungen unter anderem auch Schlei-

fenkonstrukte und weitere Kontrollstrukturen für Aktionen (siehe Action Module) zur Verfügung stellt.

Sie erreichen das eXforms-Projekt unter **http://exforms.org/**.

XForms MustUnderstand Module

Dieses Modul enthält kein Element, sondern ein Attribut. Dieses kann beliebigen Elementen zugeordnet werden, die als funktionale Erweiterung des XFormulars fungieren (z. B. in einem `<extension>`-Container). Im Fall einer funktionskritischen Erweiterung wird das `mustUnderstand`-Attribut auf `true` gesetzt: Erkennt eine Anwendung das es tragende Element nicht, soll die Verarbeitung des XFormulars abgebrochen bzw. verweigert werden.

Attribut	Rolle	für Element
`mustUnderstand`	Typ: `xsd:boolean`; Überprüfung der Verfügbarkeit einer Funktionalität	alle

Tabelle 13.6 Attribute des XForms MustUnderstand Module

Anmerkung: Da das `mustUnderstand`-Attribut regelmäßig Elementen in anderen Namenräumen zugeordnet wird, erhält es grundsätzlich ein Namensraumpräfix (beispielsweise `xforms`), der an dem XForms-Namensraum gebunden sein muss. Hier ein Beispiel für die Anmeldung einer eXforms-Erweiterung (die einen eigenen Namensraum mit Präfix `exf` verwendet):

```
...
<xforms:extension>
    <exf:requiredModules   modules=""
     xforms:mustUnderstand="true"/>
</xforms:extension>
...
```

XForms Group Module

Dieses Modul enthält ein Element, das zur Zusammenfassung von Eingabeelementen in eine funktionale Gruppe dient.

Element	Rolle	Typ
`<group>` `... </group>`	Gruppiert Eingabefelder analog zu `<html:fieldset>`; Gruppen können verschachtelt werden und Switch-Statements enthalten	Container

Tabelle 13.7 Elemente des XForms Group Module

XForms Repeat Module

Das Modul dient dazu, die Definition sich wiederholender Strukturen (auch dynamisch erzeugt) innerhalb von XFormularen zu erleichtern. Hierzu kann (über `<repeat>`) auch direkt auf Datenmodelle in Form von Listen oder Collections zugegriffen werden.

Das Modul enthält sechs Elemente.

Element	Rolle	Typ
`<copy/>`	Kopiert Werte aus einem Knoten der Dateninstanz; analog zu `<value>`, besitzt jedoch keinen eigenen Inhalt	Leeres Element
`<delete/>`	Löscht einen Datensatz aus der gleichförmigen Datensammlung (*homogeneous collection*) der Dateninstanz	Leeres Element
`<itemset>` `... </itemset>`	Generiert eine Struktur anhand eines Nodeset-Modells	Container
`<insert/>`	Fügt in eine gleichförmigen Datensammlung einen Datensatz ein (durch Klonen eines vorhandenen Datensatzes)	Leeres Element
`<repeat>` `... </repeat>`	Erzeugt (wiederholt) eine Struktur anhand eines Nodeset-Modells (Liste oder Collection)	Container
`<setindex/>`	Setzt den Index auf ein Item und macht es dadurch zum aktuellen (*current*) Item	Leeres Element

Tabelle 13.8 Elemente des XForms Repeat Module

XForms Switch Module

Das Switch-Modul bietet in Form dreier Elemente die Möglichkeit, in XFormularen die Funktionalität von Switch-Statements nachzubilden.

Element	Rolle	Typ
`<case>` `... </case>`	Enthält die alternativen Anweisungsblöcke eines Switchstatements	Container
`<switch>` `... </switch>`	Container für Switchstatement; enthält ein oder mehrere `<case>`-Elemente	Container
`<toggle />`	Einem `<case>` zugeordnete, eventgetriggerte Schaltanweisung, die dessen Zustand bestimmt	Leeres Element

Tabelle 13.9 Elemente des XForms Switch Module

13.1.5 Datentypen in XForms

In XForms werden die meisten, in XML-Schema definierten Datentypen direkt unterstützt und können eingesetzt werden. Eine Zuordnung eines Schema-Datentyps an einen Knoten der Dateninstanz ist jedoch unabdingbar.

XML-Schema-Datentypen in XForms

Primitive XML-Schema-Datentypen, die von XForms unterstützt werden:

`xsd:dateTime`, `xsd:time`, `xsd:date`, `xsd:gYearMonth`, `xsd:gYear`, `xsd:gMonthDay`, `xsd:gDay`, `xsd:gMonth`, `xsd:string`, `xsd:boolean`, `xsd:base64Binary`, `xsd:hexBinary`, `xsd:float`, `xsd:decimal`, `xsd:double`, `xsd:anyURI` und `xsd:QName`.

Nicht unterstützt wird der primitive Typ `xsd:duration`. Hierfür stellt XForms die Datentypen `xforms:dayTimeDuration` und `xforms:yearMonthDuration` zur Verfügung (siehe unten).

Abgeleitete XML-Schema-Datentypen, die von XForms unterstützt werden:

`xsd:normalizedString`, `xsd:token`, `xsd:language`, `xsd:Name`, `xsd:NCName`, `xsd:ID`, `xsd:IDREF`, `xsd:IDREFS`, `xsd:NMTOKEN`, `xsd:NMTOKENS`, `xsd:integer`, `xsd:nonPositiveInteger`, `xsd:negativeInteger`, `xsd:long`, `xsd:int`, `xsd:short`, `xsd:byte`, `xsd:nonNegativeInteger`, `xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort`, `xsd:unsignedByte`, `xsd:positiveInteger`

Eigene XForms-Datentypen

XForms kennt einige eigene Datentypen, die in XML-Schema nicht existieren:

`xforms:listItem`, `xforms:listItems`, `xforms:dayTimeDuration`, `xforms:yearMonthDuration`

Die ersten beiden dienen zur Typisierung von XForms-eigener List-Items, die beiden anderen als Ersatz für den XML-Schema-Typ `xsd:duration` – sie sind in Anlehnung an die entsprechenden XPath-eigenen Datentypen für Zeitdauerwerte implementiert.

13.1.6 XPath in XForms

XForms erlaubt in bestimmten Attributen einiger seiner Elemente den Einsatz von XPath-Ausdrücken (*expressions*), zum aktuellen Zeitpunkt noch gemäß der Spezifikation von **XPath 1.0**. Aktuelle Implementierungen von XForms sind gehalten, XPath 1.0 **in vollem Umfang** zu unterstützen (spätere XForms-Versio-

nen werden jedoch XPath 2.0 verwenden, um auch hier XML-Schema unterstützen zu können)[3].

Welche Bedeutung hat XPath für XForms?

Mit XPath-Ausdrücken kann im DOM der Dateninstanz des XFormulars (das die aktuelle Struktur der erfassten Werte in Form einer XML-Baumstruktur repräsentiert) navigiert und dort Knoten ausgewählt werden. Zur Erzeugung von Nodesets wird ein XPath-Pfadausdruck (*binding expression*) verwendet. Ausgewählte Knoten oder Knotensets werden einem XFormular-Element zugewiesen. Entsprechend ist auch die Zuweisung eines berechneten Wertes (*computed expression*) mit Hilfe von XPath-Funktionen möglich.

13.1.7 XForms-Funktionen

XForms stellt in Zusammenhang mit der Bearbeitung oder Erfassung der Dateninstanz auch eigene Funktionen und Methoden zur Verfügung bzw. bildet Funktionen aus XPath nach (hierunter befinden sich auch Funktionen, die ihre Pendants erst in Funktionen aus XPath 2.0 besitzen).

Funktionen zur DOM-Manipulation der Dateninstanz

Funktion	Bedeutung
`getInstanceDocument()`	kein Parameter; die Dateninstanz als DOM-Baumstruktur
`rebuild()`	kein Parameter; Erneuerung der Dateninstanz
`recalculate()`	kein Parameter; Neuberechnung der Dateninstanz
`revalidate()`	kein Parameter; Neuvalidierung der Dateninstanz
`refresh()`	kein Parameter; Neuladen der Dateninstanz

Tabelle 13.10 XForms-Funktionen zur DOM-Manipulation

3 Ebenso wenig wie eine befriedigende Darstellung von XForms ist in diesem Rahmen eine Darstellung von XPath möglich. Ich möchte hierfür auf mein Buch *XSLT 2.0 und XPath 2.0 – Das vollständige Handbuch* (Galileo Press) verweisen, das ebenfalls eine Einführung in XPath 1.0 und 2.0 enthält.

Funktionen in Zusammenhang mit dem Auswertungskontext

Funktion	Bedeutung
`position()`	Gibt die Position des Auswertungskontexts an; entspricht der gleichnamigen XPath-Funktion.
`last()`	Gibt das letzte Item der aktuellen Auswahl an; entspricht der gleichnamigen XPath-Funktion.
`count()`	Zählt die Items der aktuellen Auswahl; entspricht der gleichnamigen XPath-Funktion.
`id()`	Gibt das Item mit dem übergebenen ID zurück

Tabelle 13.11 XForms-Funktionen zum Auswertungskontext

Boolesche Funktionen

Funktion	Bedeutung
`boolean-from-string()`	1 Parameter (*string*); Wandelt Eingabestrings »true« oder »1« in den Booleschen Wert `true` um, ansonsten in `false`
`if()`	3 Parameter (*boolean*, *string*, *string*); Gibt, je nach boolescher Auswertung des ersten Parameters, entweder (`true`) den zweiten oder (`false`) den dritten Parameter zurück

Tabelle 13.12 Boolesche XForms-Funktionen

Funktionen zur Verarbeitung von Zahlenwerten

Funktion	Bedeutung
`avg()`	1 Parameter (*node-set*); Gibt den arithmetischen Mittelwertes der in Zahlen umgewandelten Stringwerte des übergebenen Nodesets zurück
`min()`	1 Parameter (*node-set*); Gibt den niedrigsten Wert unter den in Zahlen umgewandelten Stringwerten des übergebenen Nodesets zurück
`max()`	1 Parameter (*node-set*); Gibt den höchsten Wert unter den in Zahlen umgewandelten Stringwerten des übergebenen Nodesets zurück
`count-non-empty()`	1 Parameter (*node-set*); Gibt die Zahl der nicht-leeren Nodes des übergebenen Nodesets zurück
`index()`	1 Parameter (*string*); Erwartet den `IDREF` eines Repeats; gibt die Position des bezeichneten Repeats zurück, ansonsten `NaN`

Tabelle 13.13 XForms-Funktionen zur Zahlenverarbeitung

Funktionen zur Ermittlung von Eigenschaften

Funktion	Bedeutung
`property()`	1 Parameter (*string*); Gibt den Wert desbezeichneten XForms-Properties zurück

Tabelle 13.14 XForms-Funktionen zur Property-Erkennung

Datums- und Zeitfunktionen

Funktion	Bedeutung
`now()`	kein Parameter; Gibt die aktuelle Systemzeit als String nach dem kanonischen `xsd:dateTime`-Format zurück
`days-from-date()`	1 Parameter (*string*); Eingabestring lexikalisch `xsd:date`; gibt Zahl der Tage seit `1970-01-01` zurück, ansonsten `NaN`
`seconds-from-dateTime()`	1 Parameter (*string*); Eingabestring lexikalisch `xsd:date`; gibt Zahl der Sekunden seit `1970-01-01T00:00:00Z` zurück, ansonsten `NaN`
`seconds()`	1 Parameter (*string*); Eingabestring lexikalisch `xsd:duration`; gibt Zahl der Sekunden des Zeitdauerwertes zurück, ansonsten `NaN`
`months()`	1 Parameter (*string*); Eingabestring lexikalisch `xsd:duration`; gibt Zahl der Monate des Zeitdauerwertes zurück, ansonsten `NaN`

Tabelle 13.15 XForms-Funktionen zur Datums- und Zeitdauerverarbeitung

Funktionen für Nodesets

Funktion	Bedeutung
`instance()`	1 Parameter (*string*); Erwartet den `IDREF` einer Dateninstanz; gibt den Dokumentknoten der bezeichneten Instanz zurück, ansonsten einen leeren Nodeset

Tabelle 13.16 XForms-Funktionen zur Wahl einer Dateninstanz

13.1.8 Events in XForms

Ein Formular im Web benötigt Methoden, die Benutzerinteraktionen (*events*) aufzufangen, um auf sie adäquat reagieren zu können. Was für HTML-Formulare gilt, gilt umso mehr für XForms. Anstatt für das Eventhandling lediglich das als limitiert betrachtete DOM-Eventmodell zu implementieren, entschied man sich, zusätzlich auch auf das das leistungsfähigere und sprachübergreifende **XML-Events**-Modell zu setzen. Die Eventverarbeitung von XForms besteht dement-

sprechend nach dem Auftreten des Events (das hier nicht mitgezählt wird) aus drei Phasen:

1. Weiterreichen des Events ab Quellknoten (*event bubbling phase*)
2. Ankunft des Events am Zielknoten (*arrival of the event at its target*)
3. Eventerkennung (*event capture phase*)

Sie erreichen die W3C Recommendation für XML-Events unter **www.w3.org/TR/xml-events/**.

Namenraum für XML-Events

Das XML-Eventmodul benötigt den Namensraum **http://www.w3.org/2001/xml-events.**

Bevorzugter Präfix ist `ev`. Es sollte daher in einem Dokument, das XML-Events einsetzt, folgende **Namenraumdeklaration** vorliegen (möglichst im Wurzelelement):

```
xmlns:ev="http://www.w3.org/2001/xml-events"
```

Eventdefinitionen für XForms

XForms teilt die hier definierten Events in vier Gruppen ein:

- **Initialization Events**
  Dienen dazu, das XFormular, seine Felder und Dateninstanz zu initialisieren
- **Interaction Events**
  Dienen dazu, die Interaktion mit dem Nutzer zu ermöglichen
- **Notification Events**
  Dienen dazu den Nutzer über Zustände des XFormulars zu benachrichtigen
- **Error Indications**
  Dienen der Fehlererkennung und -behandlung

Event	Cancelbar	Bubbelt	Zielelement
`xforms-model-construct`	Nein	Ja	`<model>`
`xforms-model-construct-done`	Nein	Ja	`<model>`
`xforms-ready`	Nein	Ja	`<model>`
`xforms-model-destruct`	Nein	Ja	`<model>`

Tabelle 13.17 Initialization Events

Event	Cancelbar	Bubbelt	Zielelement
xforms-previous	Ja	Nein	div. Formelemente
xforms-next	Ja	Nein	div. Formelemente
xforms-focus	Ja	Nein	div. Formelemente
xforms-help	Ja	Ja	div. Formelemente
xforms-hint	Ja	Ja	div. Formelemente
xforms-rebuild	Ja	Ja	<model>
xforms-refresh	Ja	Ja	<model>
xforms-revalidate	Ja	Ja	<model>
xforms-recalculate	Ja	Ja	<model>
xforms-reset	Ja	Ja	<model>
xforms-submit	Ja	Ja	<submission>

Tabelle 13.18 Interaction Events

Event	Cancelbar	Bubbelt	Zielelement
DOMActivate	Nein	Ja	div. Formelemente
xforms-value-changed	Nein	Ja	div. Formelemente
xforms-select	Nein	Ja	<item>, <case>
xforms-deselect	Nein	Ja	<item>, <case>
xforms-scroll-first	Nein	Ja	<repeat>
xforms-scroll-last	Nein	Ja	<repeat>
xforms-insert	Nein	Ja	<instance>
xforms-delete	Nein	Ja	<instance>
xforms-valid	Nein	Ja	div. Formelemente
xforms-invalid	Nein	Ja	div. Formelemente
DOMFocusIn	Nein	Ja	div. Formelemente
DOMFocusOut	Nein	Ja	div. Formelemente
xforms-readonly	Nein	Ja	div. Formelemente
xforms-readwrite	Nein	Ja	div. Formelemente
xforms-required	Nein	Ja	div. Formelemente
xforms-optional	Nein	Ja	div. Formelemente
xforms-enabled	Nein	Ja	div. Formelemente
xforms-disabled	Nein	Ja	div. Formelemente
xforms-in-range	Nein	Ja	div. Formelemente
xforms-out-of-range	Nein	Ja	div. Formelemente
xforms-submit-done	Nein	Ja	<submission>
xforms-submit-error	Nein	Ja	<submission>

Tabelle 13.19 Notification Events

Event	Cancelbar	Bubbelt	Zielelement
`xforms-binding-exception`	Nein	Ja	bindbare Elemente
`xforms-link-exception`	Nein	Ja	`<model>`
`xforms-link-error`	Nein	Ja	`<model>`
`xforms-compute-exception`	Nein	Ja	`<model>`

Tabelle 13.20 Error Indications

13.1.9 CSS-Selektoren für XForms

Zum Styling eines XFormulars sollte CSS verwendet werden. Dies ist für die XForms-Elemente verhältnismäßig einfach. Für die unterschiedliche Darstellung der unterscheidbaren Zustände der Formularelemente muss jedoch auf Pseudoklassen zurückgegriffen werden. Die hierfür im Prinzip erforderlichen Selektoren sind selbst in CSS3 noch nicht vollständig enthalten. Im Rahmen der XForms-Spezifikation finden sich Vorschläge für eine entsprechende zukünftige Erweiterung.

Selektoren aus der CSS3-Spezifikation

In CSS3 sind bereits zwei Pseudoklassen vorgesehen, die in Verbindung mit XForms genutzt werden können.

- `:enabled`, `:disabled`
  Die Pseudoklassen `:enabled` und `:disabled` dienen dazu, ein Formularelement entsprechend unterschiedlich zu stylen, je nachdem, ob es aktiv (*enabled*) oder deaktiviert (*disabled*) ist. Hierfür muss das Property `relevant` mit Wert `true` oder `false` an das Element gebunden sein.

Selektoren aus XForms-Spezifikation

Weitere Selektoren werden im Rahmen der XForms-Spezifikation vorgeschlagen, sind jedoch bislang noch nicht Teil der CSS3-Spezifikationen. Sie sind zum Teil versuchsweise in X-Smiles implementiert, nicht jedoch in Firefox:

- `:valid`, `:invalid`
  Dienen dazu, ein Formularelement entsprechend unterschiedlich zu stylen, je nachdem, ob der eingegebene Wert gültig (*valid*) oder ungültig (*invalid*) gegenüber dem an das Element gebundenen Datentyp ist.
- `:in-range`, `:out-of-range`
  Dienen dazu, ein Formularelement entsprechend unterschiedlich zu stylen, je nachdem, ob der eingegebene Wert innerhalb (*in-range*) oder außerhalb *(out-of-range*) des vorgesehenen Wertebereichs ist.

- `:required`, `:optional`
  Dienen dazu, ein Formularelement entsprechend unterschiedlich zu stylen, je nachdem, ob eine Werteingabe in das Feld erforderlich (*required*) oder nicht erforderlich *(optional*) ist. Hierfür muss das Property `required` mit Wert `true` oder `false` über einen Bind an das Element gebunden sein.
- `:read-only`, `:read-write`
  Dienen dazu, ein Formularelement entsprechend unterschiedlich zu stylen, je nachdem, ob auf das Feld nur lesend (*read-only*) oder auch *schreibend (read-write*) zugegriffen werden kann. Hierfür muss das Property `readonly` mit Wert `true` oder `false` an das Element gebunden sein.

Zusätzlich sind derzeit drei **Pseudo-Element-Selektoren** definiert.

- `::value`
  Der Selektor `::value` dient dazu, den Wert eines Ein- oder Ausgabeelements zu stylen. Dies ist beispielsweise bei `<output>`-Elementen sinnvoll, da die Wertausgabe nicht innerhalb eines eigenen Elements erfolgt, sonst also nicht erfasst wird. Gestylt wird der Bereich nach dem `<label>`, das ebenfalls in das Element eingeschlossen ist.
- `::repeat-item`
  Erfasst ein Item einer Itemsequenz.
- `::repeat-index`
  Erfasst das aktuelle (*current*) Item einer Itemsequenz.

Anmerkung: Die Liste der CSS-Vorschläge im Rahmen der XForms-Spezifikation gilt ausdrücklich als noch unvollständig. Es ist damit zu rechnen, dass weitere Selektoren bis zur endgültigen Verabschiedung hinzukommen.

13.1.10 Aufbau eines XFormulars

Ein umgebendes `<form>`-Element wie in HTML ist nicht erforderlich, um ein XFormular zu definieren. Die Funktionalität wird durch ein im Dokumentkopf befindliches Element `<model>` ersetzt. Dieses Formularmodell verweist optional auf ein XML-Schema, das die Datentypen der versendeten Daten reglementiert. Zusätzlich können mittels eines Identifiers mehrere Formularmodelle unterschieden werden.

```
<xfm:model id="formular" schema="kontakt.xsd">
  ...
</xfm:model>
```

Die Unterelemente `<instance>` und `<submission>` des Formularmodells bestimmen Übertragungsart und Datenstruktur:

```
<xfm:model id="formular" schema="kontakt.xsd">

  <!-- Art des Datenversands: -->
  <xfm:submission id="submit_a" method="post" action="..."/>

  <!-- Form der Datenstruktur: -->
  <xfm:instance id="instance" xmlns="">
  ...
  </xfm:instance>

</xfm:model>
```

In `<instance>` wird die Definition der XML-Dateninstanz des Formulars vorgenommen. Das Datenmodell kann dabei einfach als XML in den Container eingefügt werden, ihre Elementnamen und Datentypen müssen in der XML-Schema-Datei definiert werden, auf die in `<model>` verwiesen wird Der Namensraum der Dateninstanz wird innerhalb des Formulars explizit genullt.

```
...
<xfm:instance id="instance" xmlns="">
        <kontakt>
            <vorname>Klaus</vorname>
            <nachname>Mustermann</nachname>
            <adresse>
               <plz>12345</plz>
               <ort>Beispielstadt</ort>
               <strasse>Testgasse</strasse>
               <nummer>7</nummer>
            </adresse>
        </kontakt>
</xfm:instance>
...
```

Alternativ zum Einfügen des Datenmodells in das Element ist auch ein Link auf eine externe XML-Ressource möglich:

```
...
<xfm:instance id="instance" xmlns="" src="kontakt.xml"/>
...
```

Das Datenmodell bestimmt die **Struktur** der in XML-Form versandten Daten. Ein von dem obigen Formular versandter Datensatz könnte nach erfolgter Eingabe daher folgendermaßen aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<kontakt xmlns:xfm="http://www.w3.org/2002/xforms">
```

```
   <vorname>Peter</vorname>
   <nachname>Pan</nachname>
   <adresse>
      <plz>00000</plz>
      <ort>Neverland</ort>
      <strasse>Feenplatz</strasse>
      <nummer>1</nummer>
    </adresse>
</kontakt>
```

Beziehung zwischen Feld und Datenmodell

Die Dateninstanz des XFormulars dient zur Aufnahme der eingegebenen Daten, aber auch zur Definition von **Vorgabewerten** für die Formularfelder. Zwischen einem Knoten der Dateninstanz und einem Eingabefeld wird eine Beziehung über das `ref`-Attribut des Eingabefeldes hergestellt. Der Wert des `ref`-Attributs ist dabei ein **XPath-Ausdruck**, der auf einen Knoten der Dateninstanz zeigt. Im einfachsten Fall ähnelt dies einer Pfadangabe.

Um beispielsweise auf den Knoten `<plz>` in folgender Dateninstanz zu zeigen, wird der XPath-Ausdruck `ref="adresse/plz"` verwendet.

```
<kontakt>
   <vorname>Klaus</vorname>
   <nachname>Mustermann</nachname>
   <adresse>
        <plz>12345</plz>
        <ort>Beispielstadt</ort>
        <strasse>Testgasse</strasse>
        <nummer>7</nummer>
   </adresse>
</kontakt>
```

Der Grund für das Übergehen des Elements `<kontakt>` in diesem Beispiel liegt darin, dass das Wurzelelement der Dateninstanz grundsätzlich bereits der **Ausgangspunkt** des XPath-Ausdrucks ist. Es braucht (und darf!) daher nicht eigens in der Pfadangabe erwähnt werden. Für die Beziehung zwischen Feldern und Knoten der Dateninstanz spielt die Benennung des Wurzelknotens der Dateninstanz folglich keine Rolle – er taucht jedoch in den versandten Daten auf.

Das Eingabefeld für die Postleitzahl könnte so aussehen:

```
<fm:input ref="adresse/plz">
  <xfm:label>Postleitzahl</xfm:label>
</xfm:input>
```

Die über das `ref`-Attribut erfolgte Verbindung zum `<plz>`-Knoten der Dateninstanzdefinition hat eine mehrfache Bedeutung. Zum einen wird der ins Formular eingegebene Wert in diesem Knoten der Dateninstanz abgelegt, zum anderen bezieht das Eingabefeld auch seinen **Vorbelegungswert** aus deren Definition. Im vorliegenden Fall wäre der Vorbelegungswert für das Postleitzahlenfeld daher `12345`.

Versteckte Felder nicht mehr erforderlich

Da im Dateninstanzmodell auch Elemente erscheinen dürfen, die nicht an Eingabefelder gebunden sind, lassen sich »Hidden Fields« ganz einfach durch Vorbelegungen im Datenmodell ersetzen, die nicht im Formular erscheinen. Entsprechende Werte sind vom Formular aus nicht veränderlich, werden jedoch beim Datenversand mitverschickt. Ein XForms-Pendant zu `<input type="hidden">` aus HTML existiert daher nicht.

Hints als Eingabehilfen

Ein Eingabefeld kann Ausfüllhilfen in Form so genannter Hints erhalten, die bei Mouseover als Tooltips sichtbar werden. Will man dem Nutzer mitteilen, dass er eine fünfstellige Ziffernfolge eingeben soll, kann dies folgendermaßen geschehen:

```
<fm:input ref="adresse/plz">
  <xfm:label>Postleitzahl</xfm:label>
  <xfm:hint>Bitte eine fünfstellige Ziffernfolge.</xfm:hint>
</xfm:input>
```

In diesem Fall ist eine Wertvorgabe nicht unbedingt sinnvoll, das Feld sollte zu Anfangs besser leer sein. Der Nutzer soll jedoch angehalten sein, wirklich eine Postleitzahl einzutragen.

13.1.11 Obligatorische und optionale Eingaben

Ein wesentliches Feature von XForms besteht in der Möglichkeit, Formularfelder als zum Ausfüllen *obligatorisch* oder als *optional* zu bezeichnen. Die Eingabe eines Wertes kann so erzwungen werden, wobei mittels XML-Schema-Constraints die Art der gewünschten Eingabe genau umrissen werden kann – ohne eine entsprechend passende Eingabe wird das Formular nicht verschickt.

Hierzu dient das `<bind>`-Element in `<model>`, das wiederum einen XPath-Ausdruck verwendet, um den entsprechenden Knoten zu lokalisieren:

```
<xfm:model id="formular" schema="form.xsd">
    <xfm:submission id="submit" method="post" action="..."/>
    <xfm:instance id="instance" xmlns="">
       <kontakt>
       ...
           <adresse>
```

```
            <plz>12345</plz>
            ...
          </adresse>
        </kontakt>
    </xfm:instance>
    <xfm:bind nodeset="adresse/plz" required="true()"/>
</xfm:model>
```

Das Eingabefeld, das an den Knoten `<plz>` der Dateninstanz gebunden ist, muss ausgefüllt werden, was durch `required="true()"` im `<bind>`-Element erreicht wird. Bleibt das Feld leer, erfolgt beim Datenversand eine Fehlermeldung.

13.1.12 Reglementierung der Eingabedaten durch XML-Schema

Noch interessanter, als ein Feld als obligatorisch zu kennzeichnen, ist die Möglichkeit, auch eine Beschränkung der einzugebenden Werte vornehmen zu können. Dies geschieht mittels des XML-Schema, das über den `<model>`-Container an die Dateninstanz gebunden ist:

```
<xfm:model id="formular" schema="kontakt.xsd">
       ...
</xfm:model>
```

Um die Eingabe in das Postleitzahlenfeld nur in Form einer fünfstelligen Ziffer zuzulassen, braucht `kontakt.xsd` nur eine Definition des Elements `<plz>` zu beinhalten, etwa in folgender Form:

```
<xs:element name="plz">
   <xs:simpleType>
      <xs:restriction base="xs:integer">
         <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Entspricht die Eingabe nicht den Forderungen des Schemas, so können die Daten nicht versendet werden. Gleichzeitig kann die Eingabe durch Styleangaben als unzulässig markiert werden. Ein XFormular bringt also alle Features, die zur Validierung von Formularen erforderlich sind, bereits von Haus aus mit, ohne dass eine zusätzliche client- oder serverseitige Programmierung, beispielsweise mit JavaScript oder PHP, erforderlich wäre. Um dies in vollem Umfang auszunutzen, sind Kenntnisse in XML-Schema natürlich von Vorteil.[4]

4 Eine Vertiefung zu XML Schema kann an dieser Stelle nicht erfolgen. Ich möchte Ihnen hierfür das einschlägige Buch *XML Schema* von Marco Skulschus und Marcus Wiederstein (Galileo Press) empfehlen, das eine sehr gut lesbare Einführung in das Thema bietet.

Auswahlen – Selectboxen, Checkboxen, Radiobuttons

In XForms dienen zwei Elemente der Erstellung von Auswahlfeldern, wobei zunächst nur unterschieden wird, ob lediglich eine der angebotenen Optionen wählbar ist (Einfachauswahl) oder mehrere (Mehrfachauswahl): Je nachdem wird das Element `<select1>` oder das Element `<select>` eingesetzt.

Als Unterelement für `<select>` und `<select1>` wird `<item>` eingesetzt, das wiederum ein optionales `<label>` tragen kann und ein Unterelement `<value>` besitzt, das den zu übertragenden Wert definiert:

```
<xfm:item>
   <xfm:label>Wasser</xfm:label>
   <xfm:value>wasser</xfm:value>
</xfm:item>
```

Die beiden XForms-Select-Elemente dienen zur Erzeugung von Selectboxen nach HTML-Vorbild, ebenfalls jedoch zur Erzeugung von Checkboxen oder Radiobutton-Arrays, für die deshalb keine eigenen Inputformen vorgesehen sind. Durch Randbedingungen wird bestimmt, auf welche Art die Auswahl durch den User Agent präsentiert werden soll. Einen wesentlichen Beitrag dazu bringt das `appearance`-Attribut:

- **Einfachauswahl**

  `<select1 appearance="compact">`
  für mehrzeiliges Selectfeld, ein Wert wählbar

  `<select1 appearance="minimal">`
  für einzeiliges Selectfeld, ein Wert wählbar

  `<select1 appearance="full">`
  für Radiobutton

- **Mehrfachauswahl**

  `<select appearance="compact">`
  für mehrzeiliges Selectfeld, mehrere Werte wählbar

  `<select appearance="minimal">`
  für einzeiliges Selectfeld, mehrere Werte wählbar

  `<select appearance="full">`
  für Checkbox

[zB] *Beispiele*:

Einfachauswahl als mehrzeiliges Selectfeld:

```
<xfm:select1 appearance="compact">
      <xfm:item>
```

```
            <xfm:label>Rotwein</xfm:label>
            <xfm:value>rot</xfm:value>
        </xfm:item>
        <xfm:item>
            <xfm:label>Weisswein</xfm:label>
            <xfm:value>weiss</xfm:value>
        </xfm:item>
</xfm:select1>
```

Einfachauswahl als einzeiliges Selectfeld:

```
<xfm:select1 appearance="minimal">
        <xfm:item>
            <xfm:label>Wasser</xfm:label>
            <xfm:value>wasser</xfm:value>
        </xfm:item>
        <xfm:item>
            <xfm:label>Coca Cola</xfm:label>
            <xfm:value>cola</xfm:value>
        </xfm:item>
</xfm:select1>
```

Einfachauswahl als Radiobutton-Array:

```
<xfm:select1 appearance="full">
        <xfm:item>
            <xfm:label>Kristallweizen</xfm:label>
            <xfm:value>kristall</xfm:value>
        </xfm:item>
        <xfm:item>
            <xfm:label>Hefeweizen</xfm:label>
            <xfm:value>hefe</xfm:value>
        </xfm:item>
</xfm:select1>
```

Mehrfachauswahl in Form von Checkboxen:

```
<xfm:select appearance="full">
        <xfm:item>
            <xfm:label>Erdbeereis</xfm:label>
            <xfm:value>erdbeer</xfm:value>
        </xfm:item>
        <xfm:item>
            <xfm:label>Vanilleeis</xfm:label>
            <xfm:value>vanille</xfm:value>
        </xfm:item>
</xfm:select>
```

Abbildung 13.1 Variable Präsentation von Auswahlfeldern

Die Ausprägung und Funktion der Felder hängt mittelbar auch mit dem an sie jeweils gebundenen XML-Schema-Datentyp zusammen. Eine Checkbox benötigt in X-Smiles notwendigerweise z. B. den Typ `xsd:boolean`.

Noch besteht keine Einigkeit zwischen den User Agents, wie Elemente dargestellt werden sollen. X-Smiles stellt beispielsweise Radiobutton-Arrays unrichtigerweise als Combo-Box dar, während die Mozilla XForms-Erweiterung von Firefox hier korrekte Radiobuttons zeigt. Andererseits erschwert die für XForms-Zwecke noch unzureichende CSS-Unterstützung in Firefox (es werden die Pseudo-Selektoren `input::value` oder `output::value` nicht interpretiert) die gewünschte Präsentation des XFormulars.

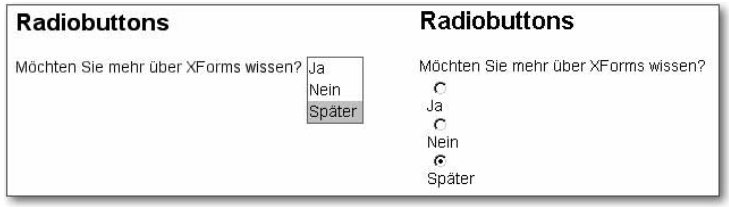

Abbildung 13.2 XForms-Radiobuttons in X-Smiles (links) und Firefox (rechts)

Gruppierungen von Items eines Auswahlfeldes

Innerhalb einer Liste von Items, die als Auswahlmöglichkeiten dienen, können per `<xfm:choices>`-Element Gruppierungen definiert werden, die durch ein eigenes, unmittelbar innerhalb liegendes `<xfm:label>`-Element betitelt werden.

Mit folgender Selectbox kann genau ein Gericht aus den Bereichen »Pizza« oder »Salat« gewählt werden.

```
<xfm:select1 ref="bestellung/essen">
  <xfm:label>Speisekarte:</xfm:label>

    <xfm:choices>
    <xfm:label>Pizza</xfm:label>
      <xfm:item>
          <xfm:label>Pizza Margerita</xfm:label>
          <xfm:value>p_mar</xfm:value>
```

```
      </xfm:item>
      <xfm:item>
          <xfm:label>Pizza Tonno</xfm:label>
          <xfm:value>p_ton</xfm:value>
      </xfm:item>
      <xfm:item>
          <xfm:label>Pizza Calzone</xfm:label>
          <xfm:value>p_cal</xfm:value>
      </xfm:item>
    </xfm:choices>

    <xfm:choices>
    <xfm:label>Insalata</xfm:label>
      <xfm:item>
          <xfm:label>Insalata Mista</xfm:label>
          <xfm:value>i_mis</xfm:value>
      </xfm:item>
      <xfm:item>
          <xfm:label>Insalata Regina</xfm:label>
          <xfm:value>i_reg</xfm:value>
      </xfm:item>
      <xfm:item>
          <xfm:label>Insalata Tonno</xfm:label>
          <xfm:value>i_ton</xfm:value>
      </xfm:item>
    </xfm:choices>

</xfm:select1>
```

Die Wirkung der `<choices>`-Elemente entspricht weitestgehend derjenigen der `<optgroup>`-Elemente aus HTML. Die Umsetzung ist in den beiden getesteten User Agents nicht identisch – in Firefox ist sie näher am Gewünschten (X-Smiles stellt die Label der Gruppierung nicht innerhalb der Auswahlbox dar).

Abbildung 13.3 XForms Selectbox in Firefox

13.1.13 Versenden der Formulardaten – Der Submit-Button

Auch ein XFormular benötigt eine Submit-Funktion, die allerdings flexibler ausgestaltet ist als ihr HTML-Pendant. Anstelle von `<html:input type="submit">` wird ein spezifischeres Element `<submit>` eingesetzt, das ebenso wie die `<input>`-Elemente von XForms als Container ausgelegt ist. Es kann also sowohl `<label>` als auch `<hint>`-Elemente enthalten, wobei der Inhalt des Labels als Buttonaufschrift dient:

```
<xfm:submit ...>
   <xfm:label>Kontaktdaten versenden</xfm:label>
   <xfm:hint>Klicken Sie hier,
   um die Daten zu versenden</xfm:hint>
<xfm:submit>
```

Da es keinen alle Formularelemente umgebenden Container gibt, muss die Beziehung zwischen der Formularmodellierung und dem Submit-Button wieder über eine `IDREF`-Beziehung erfolgen. Ziel eines Submits ist das `<submission>`-Element im `<model>`-Container:

```
<!-- Im Dokumentkopf: -->
<xfm:model ...>
...
 <xfm:submission id="submit_a" method="post" action="..."/>
</xfm:model>

<!-- Im Dokumentkörper: -->
<xfm:submit submission="submit_a">
   ...
<xfm:submit>
```

Übertragungsmethoden für XForms

Die Definition der Übertragungsmethode, die in HTML-Formularen zwei Attribute erfordert, nämlich `method` und `enctype`, ist in XForms vereinfacht – hier ist lediglich ein `method`-Attribut im `<submission>`-Element erforderlich.

Die Möglichkeiten wachsen dennoch auf sechs Übertragungsmethoden an:

XForms-Methode	Serialisiert als	HTML-Äquivalent
`method="get"`	`application/x-www-form-urlencoded`	`method="get"`
`method="urlencoded-post"`	`application/x-www-form-urlencoded`	`method="post" enctype="application/x-www-form-urlencoded"`

XForms-Methode	Serialisiert als	HTML-Äquivalent
`method="multipart-post"`	`multipart/related`	`multipart/related`
`method="form-data-post"`	`multipart/form-data`	`method="post" enctype="multipart/form-data"`
`method="post"`	`application/xml`	– (nicht vorhanden)
`method="put"`	`application/xml`	– (nicht vorhanden)

Zu beachten ist, dass `method="post"` nicht identisch ist mit `method="post"` in HTML, da dort ohne Nennung von `enctype` automatisch `enctype="text/plain"` angenommen wird. In XForms werden die Formulardaten jedoch automatisch als XML übertragen, sofern `method="post"` gewählt ist.

Eine für browserähnliche Clients vollkommen neue Übertragungsmethode ist `method="put"`, die entsprechend der HTTP-Methode `PUT` die Formularwerte als XML-Datei lokal oder auf einem Server ablegen oder Teile einer vorhandenen Datei ändern kann. Hierbei wird der Name der abzulegenden Datei per `action`-Attribut festgelegt. In einer bestehenden Ziel-Datei zu ändernde Knoten werden über einen XPath-Zeiger im `ref`-Attribut des Feldes bestimmt, das den jeweiligen Wert zur Verfügung stellt. Auf diesem Wege kann quasi über ein XForms-Formular eine auf den Server befindliche Datei editiert werden.

Eine `put`-Variante besteht in der Ablage der Formulardaten in XML-Form im lokalen File-System. Hierzu wird bei `action` das File-Protokoll mit Präfix `file:` angesprochen.

```
<!-- im Dokumentkopf: -->
<model>
...
<submission action="file:results.xml"
 method="put" id="save"/>
...
</model>

<!-- im Dokument: -->
<submit submission="save">Lokal speichern</submit>
```

Mittels alternativer Submit-Buttons kann der Nutzer beispielsweise wählen, ob er Formularinhalte speichern oder verschicken will.

13.1.14 Mehrfaches Submit

Anstatt wie bislang nur an einen festen Empfänger, kann der Inhalt des XForms-Formulars wahlweise an verschiedene Server geschickt werden (allerdings nicht

an alle gleichzeitig). Hierzu ist es lediglich erforderlich, mehrere <submission>-Elemente zu definieren, denen jeweils ein <submit>-Element zugeordnet ist. Auch hierzu dient wieder eine ID-IDREF-Beziehung.

```
<xfm:model>
   <xfm:instance>
   ...
   <xfm:submission  action="..."  method="get"  id="a"/>
   <xfm:submission  action="..."  method="get"  id="b"/>
</xfm:model>

<xfm:submit submission="a">
   <xfm:label>Daten an A schicken</xfm:label>
</xfm:submit>
<xfm:submit submission="b">
   <xfm:label>Daten an B schicken</xfm:label>
</xfm:submit>
```

13.1.15 Mehrere Formulare

In einem Dokument können mehrere, voneinander unabhängige XFormulare untergebracht werden. Hierfür müssen mehrere Instanzen von <model>-Containern im Dokumentkopf untergebracht werden und diese durch Identifier gekennzeichnet werden. Dies geschieht mittels des allgemeinen Attributs id:

```
<xfm:model id="form1">
   ...
</xfm:model>

<xfm:model id="form2">
   ...
</xfm:model>
```

Der Bezug zwischen den Eingabefeldern und ihrem Formularmodell wird einzeln über eine IDREF-Beziehung deklariert:

```
<xfm:input model="form1" ...>
   <xfm:label>Eingabefeld Formular 1:</xfm:label>
</xfm:input>

<xfm:input model="form2" ...>
   <xfm:label>Eingabefeld Formular 2:</xfm:label>
</xfm:input>
```

Die Bindungen eines Eingabeelements mittels seines ref-Attributs beziehen sich auf die Dateninstanz jenes Formularmodells, auf das sich das model-Attribut bezieht.

Die Angabe des Formularmodells mit dem `model`-Attribut ist nicht nötig, wenn nur ein Formular im Dokument enthalten ist. Besitzt ein Eingabefeld kein `model`-Attribut, befinden sich im Dokumentkopf jedoch mehrere Formularmodelle, so gehört das Eingabefeld zum ersten Formularmodell in der Dokumentreihenfolge.

13.1.16 Beziehung zwischen Feldern und Dateninstanz

Die in das Eingabefeld eines XFormulars eingetragenen Werte werden unmittelbar in die Dateninstanz weitergegeben, genauer gesagt, in dem sie dort repräsentierenden Knoten abgelegt. Die Dateninstanz existiert als abstraktes XML-Fragment (*abstract instance data*) – die in ihr abgelegten Werte sind jedoch konkret und für andere Formularelemente zugänglich. Eine einfache Dateninstanz könnte folgendermaßen aussehen:

```
<xfm:instance id="instance1" xmlns="">
    <eingabe>
        <text/>
    </eingabe>
</xfm:instance>
```

Die Beziehung zwischen den Eingabefeldern und der Definition der Dateninstanz geschieht, wie bereits erwähnt, über das `ref`-Attribut der Inputs. Dieses enthält einen XPath-Ausdruck, der am Wurzelelement der Dateninstanz (hier bei `<eingabe>`) ansetzt und von dort ins Innere der Definition zeigt:

```
<xfm:input model="form1" ref="text" ... >
   ...
</xfm:input>
```

Hier wird lediglich die XPath-Pfadangabe `text` benötigt, da der Zielknoten unmittelbares Kindelement von `<eingabe>` ist. Das Feld legt seinen Wert im bezeichneten Knoten der Dateninstanz ab.

Existiert ein zweites Eingabeelement mit gleichem `ref`-Attributwert, so steht es in Beziehung zum gleichen Knoten der Dateninstanz. Diese Beziehung ist keine Einbahnstraße: Ebenso wie ein Feld dort einen Wert ablegen kann, kann es einen Wert der Dateninstanz auslesen – ähnliches geschieht ja bei der Vorgabe von Werten innerhalb der Dateninstanz, die beim Aufrufen des Formulars als Ausgangswerte in den Feldern erscheinen.

Das Auslesen der Werte der Dateninstanz ist jedoch auch dynamisch möglich. Auf diesem Wege kann eine (eigentlich indirekte) Beziehung zwischen zwei Eingabefeldern aufgebaut werden:

```
<xfm:input model="form1" ref="text" incremental="true">
    <xfm:label>Eingabefeld 1:</xfm:label>
</xfm:input>

<xfm:input model="form1" ref="text" incremental="true">
    <xfm:label>Eingabefeld 2:</xfm:label>
</xfm:input>
```

Jedes der beiden Felder kann lesend wie schreibend auf die Dateninstanz zugreifen. Eine Eingabe in Feld 1 wird daher in Feld 2 ausgegeben. Die Angabe des Attributs `incremental="true"` bewirkt den kontinuierlichen Update der Dateninstanz.

Abbildung 13.4 Spiegelung von Eingabewerten über die Dateninstanz

Die XForms-Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:xfm="http://www.w3.org/2002/xforms">
<head>
  <title>XForms: Ein- und Ausgabeelemente</title>
  <link rel="stylesheet" type="text/css" href="xform.css" />
  <xfm:model id="form1">
    <xfm:instance id="instance1" xmlns="">
        <eingabe>
              <text/>
        </eingabe>
    </xfm:instance>
  </xfm:model>
</head>
```

```
<body>
<h1>Eingabe und Ausgabe</h1>
<p>Die im ersten Feld vorgenommenen Eingaben erscheinen unmittelbar
im zweiten Eingabefeld:</p>
<p>
<xfm:input model="form1" ref="text" incremental="true">
    <xfm:label>Eingabefeld 1:</xfm:label>
    <xfm:hint>Ein Texteingabefeld</xfm:hint>
</xfm:input>
</p>
<p>
<xfm:input model="form1" ref="text" incremental="true">
    <xfm:label>Eingabefeld 2:</xfm:label>
    <xfm:hint>Noch ein Texteingabefeld</xfm:hint>
</xfm:input>
</p>
</body>
</html>
```

13.1.17 Beispiel 1: Ein- und Ausgabefelder in XFormularen

Im folgenden Beispiel sind im Rahmen eines XFormulars Eingabefelder als Textfelder sowie Radiobuttons und Checkboxen realisiert. Die Eingaben werden unmittelbar in Ausgabefelder ausgegeben. Die Abbildung zeigt die Ausgabe in X-Smiles, das die Radiobuttons unrichtig präsentiert.

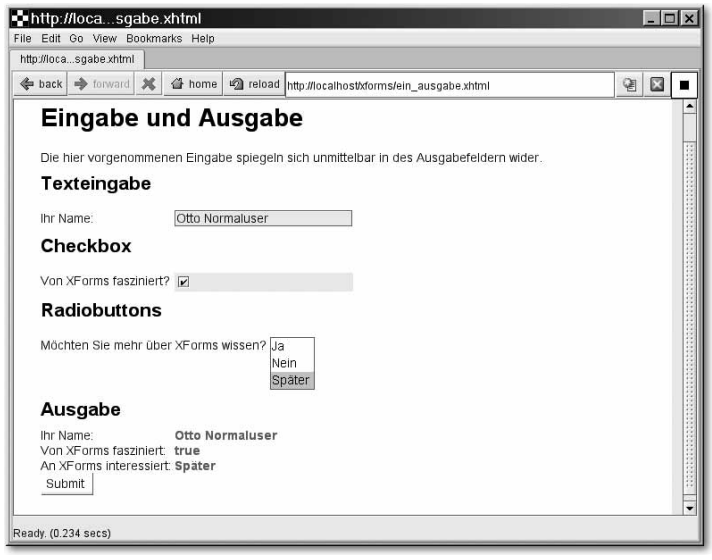

Abbildung 13.5 Ein einfaches XFormular in X-Smiles

Die XForms-Datei

Im Formularmodell wird das erforderliche XML-Schema an die Dateninstanz gebunden.

```
<xfm:model id="form1" schema="ein_ausgabe.xsd">
   ...
</xfm:model>
```

In diesem Fall ist dies erforderlich, da die Checkbox-Eingabe einen Datentyp `xsd:boolean` besitzen muss. Festgelegt wird hier im Rahmen des `<submission>`-Elements, dass die Daten lokal gespeichert werden sollen, wohin dies geschehen soll und unter welchem Dateinamen:

```
<xfm:submission id="submit1" method="put"
     localfile="C:\xampp\htdocs\xforms\ein_ausgabe.xml"/>
```

Anmerkung: Wenn Sie dieses Beispiel nachvollziehen, sollten Sie den obigen Pfad an die Verhältnisse auf Ihrem Rechner anpassen. Im Rahmen des `<instance>`-Elements wird die Datenausgabe definiert – man kann dies als »Modell« der übertragenen XML-Daten verstehen. Das XML-Schema ist an diese XML-Daten gebunden und bezieht sich auf die hier verwendeten Elementnamen. Beachten Sie die Zuweisung des leeren Namensraums **`xmlns=""`**: Die **XML-Dateninstanz** befindet sich in *keinem* Namensraum, also weder im Namensraum des XHTML-Dokuments noch im XForms-Namensraum.

```
<xfm:instance id="instance1">
   <eingabe xmlns="">
       <text>Ihr Name hier</text>
       <fasziniert>false</fasziniert>
       <auswahl></auswahl>
   </eingabe>
</xfm:instance>
```

Anstatt die Werte eines Eingabefelds an ein gleichartiges Feld zu übertragen, können diese während des Eingabeprozesses auch an ein spezielles Ausgabefeld weitergegeben und dargestellt werden. Die Methode gleicht derjenigen der Wertübergabe zwischen gleichartigen Eingabefeldern – es muss lediglich der gleiche Knoten der Dateninstanz referenziert werden. In diesem Fall entnimmt das Ausgabefeld wiederum seinen Wert dem Knoten der Dateninstanz, der durch den XPath-Ausdruck seines `ref`-Attributs bestimmt wird (es sind also nicht etwa Input und Output verknüpft):

```
<xfm:output model="form1" ref="text" ...">
   ...
</xfm:output>
```

Eingabefelder mit Auswahlmöglichkeit wie Radiobuttons und Checkboxen werden pauschal durch die beiden Elemente `<xfm:select1>` (genau ein Element wählbar) und `<xfm:select>` (kein oder beliebig viele Elemente wählbar) ersetzt, wobei die Darstellung mittels des `appearance`-Attributs bestimmt wird.

- **Checkbox**
  `<xfm:select appearance="full"> ... </xfm:select>`
- **Radiobutton**
  `<xfm:select1 appearance="full"> ... </xfm:select1>`

Der Submit-Vorgang schließlich wird durch Klick auf den **Submit**-Button eingeleitet. Dieser ist über eine `ID-IDREF`-Beziehung mit dem `<submission>`-Element der Formularmodellierung verbunden:

```
<!-- in der Formularmodellierung: -->
<xfm:submission id="submit1" ... />

<!-- Der Submit-Button: -->
<xfm:submit name="Submit" submission="submit1">
   ...
</xfm:submit>
```

Hier ein Blick auf die Datei in Ihrer Gesamtheit. Sie finden auch dieses Dokument auf der Begleit-CD:

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:xfm="http://www.w3.org/2002/xforms" xml:lang="fi">
<head>
  <title>XForms: Ein- und Ausgabeelemente</title>
  <link rel="stylesheet" type="text/css" href="xform.css" />
  <xfm:model id="form1" schema="ein_ausgabe.xsd">
    <xfm:submission id="submit1" method="put"
     localfile="C:\xampp\htdocs\xforms\ein_ausgabe.xml"/>
    <xfm:instance id="instance1" >
      <eingabe xmlns="">
          <text>Ihr Name hier</text>
          <fasziniert>false</fasziniert>
          <auswahl></auswahl>
      </eingabe>
    </xfm:instance>
    <xfm:bind nodeset="text" required="true()"/>
  </xfm:model>
</head>
<body>
<h1>Eingabe und Ausgabe</h1>
<p>Die hier vorgenommenen Eingaben spiegeln sich unmittelbar in den
Ausgabefeldern wider.</p>
```

```
<h2>Texteingabe</h2>
<p>
<xfm:input model="form1" ref="text" incremental="true"
inputmode="test">
<xfm:label>Ihr Name:</xfm:label>
<xfm:hint>Dies ist ein Texteingabefeld.</xfm:hint>
</xfm:input>
</p>

<h2>Checkbox</h2>
<p>
<xfm:input model="form1" ref="fasziniert">
<xfm:label>Von XForms fasziniert?</xfm:label>

<xfm:hint>Dies ist eine Checkbox.</xfm:hint>
</xfm:input>
</p>

<h2>Radiobuttons</h2>
<p>Möchten Sie mehr über XForms wissen?
<xfm:select1 appearance="full" ref="auswahl">
   <xfm:item>
      <xfm:label>Ja</xfm:label>
      <xfm:value>Ja</xfm:value>
   </xfm:item>
   <xfm:item>
      <xfm:label>Nein</xfm:label>
      <xfm:value>Nein</xfm:value>
   </xfm:item>
   <xfm:item>
      <xfm:label>Später</xfm:label>
      <xfm:value>Später</xfm:value>
   </xfm:item>
</xfm:select1>
</p>

<h2>Ausgabe</h2>
<xfm:output model="form1" ref="text"
 incremental="true" inputmode="test">
   <xfm:label>Ihr Name: </xfm:label>
   <xfm:hint>Ausgabe der Texteingabe</xfm:hint>
</xfm:output>

<xfm:output model="form1" ref="fasziniert">
   <xfm:label>Von XForms fasziniert: </xfm:label>
```

```
   <xfm:hint> Ausgabe des Checkboxwerts</xfm:hint>
</xfm:output>

<xfm:output model="form1" ref="auswahl">
   <xfm:label>An XForms interessiert: </xfm:label>
   <xfm:hint> Ausgabe des Radiobuttonwerts</xfm:hint>
</xfm:output>

<!-- Submit-Button -->
<xfm:submit name="Submit" submission="submit1">
   <xfm:hint>Klicken zum Versenden</xfm:hint>
   <xfm:label>Submit</xfm:label>
</xfm:submit>

</body>
</html>
```

Listing 13.1 ein_ausgabe.xhtml

Die XML-Schema-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="eingabe">
    <xsd:complexType>
       <xsd:sequence>
         <xsd:element name="text" type="xsd:string"/>
         <xsd:element name="fasziniert" type="xsd:boolean"/>
         <xsd:element name="auswahl" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 13.2 ein_ausgabe.xsd

Die versandte XML- Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<eingabe xmlns:xfm="http://www.w3.org/2002/xforms">
   <text>Otto Normaluser</text>
   <fasziniert>true</fasziniert>
   <auswahl>Später</auswahl>
</eingabe>
```

Listing 13.3 ein_ausgabe.xml

13.1.18 Beispiel 2: Eingabe eines Wertes innerhalb eines Wertebereichs

Mittels des `<xfm:range>`-Elements kann eine Eingabe auf einen definierbaren Wertebereich beschränkt werden. Das Eingabeelement nimmt die Form eines Schiebereglers an, dessen Ausgangsstellung durch den Vorbelegungswert bestimmt ist. Das Reglerelement kann entweder per Drag am Stellelement oder durch Klick in die Werteskala bedient werden.

Abbildung 13.6 XForm für Bereichseingabe in X-Smiles

Die XForms-Datei

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:xfm="http://www.w3.org/2002/xforms">
<head>
  <title>XForms: Wertebereich mit Schieberegler</title>
  <link rel="stylesheet" type="text/css" href="xform.css"/>
  <xfm:model id="formular" schema="regler.xsd">
    <xfm:submission id="submit1" method="put"
     localfile="C:\xampp\htdocs\xforms\bereich.xml"/>
    <xfm:instance id="instance1" >
        <bereichseingabe xmlns="">
            <bereich>7</bereich>
        </bereichseingabe>
    </xfm:instance>
```

```
  </xfm:model>
</head>
<body>
<h1>Wertebereich mit Schieberegler</h1>
<p>Für die Eingabe eines Wertes innerhalb eines vordefinierten Berei
chs steht ein Schieberegler zur Verfügung.</p>

<h2>Eingabefeld</h2>
<p>Hier ein Schieberegler zur Eingabe eines Wertes:<br/>
<xfm:range model="formular"  ref="bereich"
 start="-10"  end="10" step="1">
   <xfm:label>Ein Wert von -10 bis +10:</xfm:label>
   <xfm:hint>Wird als Schieberegler dargestellt.</xfm:hint>
</xfm:range></p>

<h2>Ausgabefeld</h2>
<p>
<xfm:output model="formular" ref="bereich">
   <xfm:label>Reglereinstellung: </xfm:label>
   <xfm:hint>Hier wird ein String ausgegeben.</xfm:hint>
</xfm:output></p>

<!-- Submitbutton: -->
<p>
<xfm:submit name="Submit" submission="submit1">
   <xfm:hint>Formular verschicken</xfm:hint>
   <xfm:label>Submit</xfm:label>
</xfm:submit></p>
</body>
</html>
```

Listing 13.4 regler.xhtml

Die XML-Schema-Datei

Die XML-Schema-Datei bindet über den Elementnamen des Vorgabewertes `<bereich>` in `<xfm:instance>` den Datentyp `xsd:decimal` an das Eingabefeld.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="bereichseingabe">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="bereich" type="xsd:decimal"/>
         </xsd:sequence>
      </xsd:complexType>
```

```
   </xsd:element>
</xsd:schema>
```

Listing 13.5 regler.xsd

Die versandte XML- Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bereichseingabe xmlns:xfm="http://www.w3.org/2002/xforms">
   <bereich>-4</bereich>
</bereichseingabe>
```

Listing 13.6 bereich.xml

13.1.19 Beispiel 3: Eingabe eines Datums per XForm

Interessant ist neben den Schiebereglern für die intuitive Eingabe innerhalb eines Wertebereichs auch die vereinfachte Eingabe eines Kalenderdatums mittels eines speziell angepassten Eingabefeldes `<xfm:input>`. Der Klick in ein mit dem Typ `xsd:date` belegtes Eingabefeld öffnet einen Kalender, in dem die Eingabe vorgenommen wird, die anschließend ins Eingabefeld übernommen wird (die Art der Präsentation des Kalenders variiert je nach User Agent).

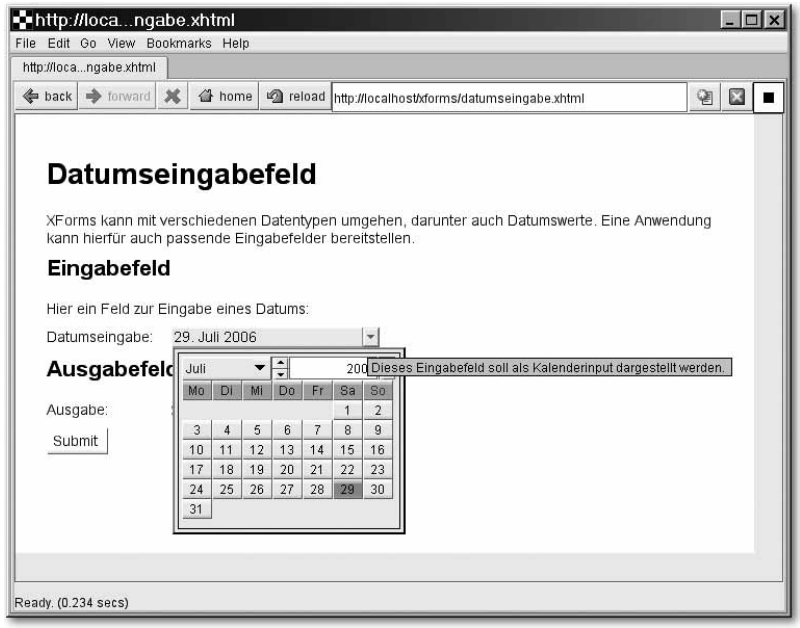

Abbildung 13.7 XForm für Datumseingabe in X-Smiles

Die XForms-Datei

Dieses XFormular nimmt einen Datumswert entgegen, wiederholt diesen in einem Ausgabefeld und speichert die Eingabe bei Submit lokal in einer XML-Datei. Innerhalb des `<xfm:model>`-Containers wird der Vorgabewert und die Datenstruktur definiert.

Auch hier gilt wieder: Korrekte Datenbindung des Eingabefelds ist essentiell. Es muss auf ein XML-Schema (hier `datum.xsd`) verwiesen werden, das explizit den Datentyp des Datumseingabefeldes auf `xsd:date` setzt – unterlässt man dies, kann der User Agent nicht die gewünschte Eingabefeldvariante liefern (in diesem Fall einen Kalenderinput).

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xfm="http://www.w3.org/2002/xforms">
<head>
  <title>XForms: Datumseingabe per Kalender</title>
  <link rel="stylesheet" type="text/css" href="xform.css" />
  <xfm:model id="formular" schema="datum.xsd">
    <xfm:submission id="submit1" method="put"
    localfile="C:\xampp\htdocs\xforms\datum.xml"/>
    <xfm:instance id="instance1" >
        <datumseingabe xmlns="">
            <datum>2006-07-21</datum>
        </datumseingabe>
    </xfm:instance>
  </xfm:model>
</head>
<body>
<h1>Datumseingabefeld</h1>
<p>XForms kann mit verschiedenen Datentypen umgehen, darunter auch
Datumswerte. Eine Anwendung kann hierfür auch passende Eingabefelder
bereitstellen.</p>

<h2>Eingabefeld</h2>
<p>Hier ein Feld zur Eingabe eines Datums:<br/>
<xfm:input model="formular" ref="datum" >
   <xfm:label>Datumseingabe:</xfm:label>
   <xfm:hint>Dieses Eingabefeld soll als Kalenderinput
   dargestellt werden.</xfm:hint>
</xfm:input></p>

<h2>Ausgabefeld</h2>
<p>
<xfm:output model="formular" ref="datum" >
   <xfm:label>Ausgabe: </xfm:label>
```

```
   <xfm:hint>Hier wird das Datum als String
   ausgegeben.</xfm:hint>
</xfm:output></p>

<!-- Submitbutton: -->
<p>
<xfm:submit name="Submit" submission="submit1">
   <xfm:hint>Formular verschicken</xfm:hint>
   <xfm:label>Submit</xfm:label>
</xfm:submit>
</p>
</body>
</html>
```

Listing 13.7 datumseingabe.xhtml

Die XML-Schema-Datei

Hier wird der Typ `xsd:date` an das Element `<datum>` gebunden.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="datumseingabe">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="datum" type="xsd:date"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Listing 13.8 datum.xsd

Die versandte XML-Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<datumseingabe xmlns:xfm="http://www.w3.org/2002/xforms">
   <datum>2006-07-27</datum>
</datumseingabe>
```

Listing 13.9 datum.xml

XForms-Beispiele bei Mozilla.org

Bei Mozilla.org finden sich eine Reihe interessanter Beispiele für XForms, die jedoch auf die Firefox-Erweiterung hin optimiert sind. In X-Smiles sind sie jedoch nicht oder nur eingeschränkt lauffähig (**http://www.mozilla.org/projects/xforms/samples.html**).

13.2 XFrames

Das Framemodell aus HTML hat mit XHTML 1.1, das ja nur noch in einer *strict*-Variante existiert, endgültig ausgedient – die Elemente `<frameset>`, `<frame>` und `<iframe>` sind im Tagvokabular nicht mehr enthalten.

Die Funktionalität von Frames, d.h. die Formulierung von Dokumenten, die das Browserfenster in Bereiche aufteilen und ihnen Inhaltsdokumente zuweisen, soll stattdessen mittels einer XML-Anwendung, einem so genannten **XFrames-Dokument** geschehen, und nicht mehr wie bisher üblich in einem HTML-Dokument.

Ein Vorteil des XFrames-Ansatzes besteht darin, dass man eine XFrames-Konfiguration als Gesamtheit in den Bookmarks ablegen kann, da die Information über die darzustellenden Ressourcen Teil des URI-Strings sind. Zudem wird eine flexiblere Einteilung des verfügbaren Darstellungsbereiches ermöglicht. Über die herkömmliche Einteilung in Zeilen und Spalten hinaus sind positionierbare sowie verschiebbare Framebereiche definierbar oder Konfigurationen, deren Steuerung der Karteireiter-Metapher folgt (*tabbed frames*).

Die Syntax solcher Dokumente ist in der **XFrames-Spezifikation** dargelegt, die zum Zeitpunkt des Erscheinens dieses Buches noch als Working Draft vorliegt, mit Stand vom 12. Oktober 2005.

W3C-Ressource

Die aktuelle XFrames-Spezifikation finden Sie unter **http://www.w3.org/TR/xframes**.

13.2.1 Die Elemente von XFrames

Das Tagvokabular von XFrames ist übersichtlich und besteht lediglich aus den sechs Elementen `<frames>`, `<head>`, `<title>`, `<style>`, `<group>` und `<frame>`, die sich, wie bei XML-Anwendungen üblich, in einem eigenen Namensraum befinden: **http://www.w3.org/2002/06/xframes/**[5].

Als allgemeiner, für alle XFrames-Elemente gültiger Attributsatz sind die drei Attribute `xml:id`, `class` und `title` definiert.

`xml:id = ID`

Das `xml:id`-Attribut weist einem Element einen eindeutigen, für das XFrames-Dokument eindeutigen Identifier zu, der zur Verknüpfung des Elements mit einer Styleanweisung oder als Linkziel dienen kann.

5 Dieser Namensraum ist im Rahmen des aktuellen Working Drafts definiert und sollte vorsichtshalber als »vorläufig« betrachtet werden.

[»] *Hinweis:* Der XML-Namensraum `xmlns:xml` ist vordefiniert und muss dem Dokument nicht explizit hinzugefügt werden.

```
class = "klassenbezeichner1 [klassenbezeichner2 ...
klassenbezeichner_n]"
```

Das `class`-Attribut weist einem Element eine oder mehrere, durch Leerzeichen getrennte CSS-Klassen zu.

```
title = "Information über das Element"
```

Das `title`-Attribut bietet Informationen über Inhalt und Bedeutung eines Xframes-Elements. Es richtet sich zur Erläuterung der XFrames-Struktur zum Zwecke der Barrierefreiheit hauptsächlich an nicht-visuelle User Agents.

Das frames-Element

Das Element `<frames>` ist das **Wurzelelement** eines XFrames-Dokuments. Es trägt auch in seinem `xmlns`-Attribut den XFrames-Namensraum. Das Inhaltsmodell besteht aus einem optionalen `<head>`-Element, gefolgt von alternativ genau einem `<group>`-Element oder einem oder mehreren `<frame>`-Elementen.

```
<frames xmlns="http://www.w3.org/2002/06/xframes/">
...
</frames>
```

Attribute:

Es gilt der Standardattributsatz mit `xml:id`, `class` und `title` und die elementspezifischen Attribute `xmlns` (obligatorisch) und `xml:base`.

```
xmlns = "http://www.w3.org/2002/06/xframes/"
```

Obligatorisch: Der Namensraum des XFrames-Dokument.

```
xml:base = Basis-URI
```

Optional: Dem `<frames>`-Element kann mit Hilfe des `xml:base`-Attributs ein Basis-URI zugewiesen werden, der zum Auflösen relativer URIs der enthaltenen Frames dient.

Das head-Element

Das `<head>`-Element enthält analog zu seinem HTML-Namensvetter die Metainformationen des XFrames-Dokument, insbesondere das obligatorische `<title>`-Element und eine beliebige Anzahl von `<style>`-Elementen. Die XFrames-DTD schreibt das Vorhandensein eines `<head>`-Containers allerdings nicht zwingend vor. Das Konzept von Metatags existiert im Rahmen der XFrames-Spezifikation (derzeit) nicht.

```
<frames xmlns="http://www.w3.org/2002/06/xframes/">
   <head>
   ...
   </head>
...
<frames>
```

Attribute:

Es gilt lediglich der Standardattributsatz mit `xml:id`, `class` und `title`.

Das title-Element

Das obligatorische `<title>`-Element enthält den Dokumenttitel des XFrames-Dokuments. Es darf nur Zeichendaten enthalten (`PCDATA`).

```
<frames xmlns="http://www.w3.org/2002/06/xframes/">
   <head>
      <title>Ein XFrames-Beispiel</title>
...
   </head>
...
</frames>
```

Attribute:

Es gilt lediglich der Standardattributsatz mit `xml:id`, `class` und `title`.

Das style-Element

Das `<style>`-Element definiert in Form von CSS-Anweisungen die Abmessungen und Darstellung der Framebereiche des XFrames-Dokuments. Sein Inhalt ist mit `PCDATA` definiert. Der Dokumentkopf eines XFrames-Dokuments darf mehrere Style-Container enthalten, die sich beispielsweise durch ihr `media`-Attribut unterscheiden. Dies ermöglicht unterschiedliche Darstellung des XFramesets je nach User Agent (Bildschirm, Projektion, Drucker etc.).

```
...
<style type="text/css" media="screen">
         #kopfbereich {height: 150px}
         #nav {width: 20%}
         #inhalt {width: 80%}
         #fussbereich {height: 4em }
</style>
...
```

Anmerkung: Die ID-Selektoren der Styledefinitionen beziehen sich auf entsprechende `xml:id` der `<frame>`-Elemente, beispielsweise also

```
...
<frame xml:id="kopfbereich" />
...
```

Attribute:

Es gilt der Standardattributsatz mit `xml:id`, `class` und `title` sowie die elementspezifischen Attribute `type` (obligatorisch), `source` und `media`.

`type = "text/css"`

Obligatorisch: Das `type`-Attribut bezeichnet den Inhaltstyp der im Element enthaltenen Styleanweisungen. Derzeit ist hierfür nur CSS vorgesehen, der gängige Attributwert lautet daher `"text/css"`. Das Attribut *muss* explizit gesetzt werden; ein Defaultwert ist nicht vorgesehen.

`source = URI-Referenz`

Optional: Das `source`-Attribut kann dem `<style>`-Element eine externe CSS-Ressource per URI zuweisen. Sofern das Laden dieser Ressource erfolgreich ist, werdem CSS-Anweisungen innerhalb des `<style>`-Containers nicht beachtet.

`media = "medientyp1 [, medientyp2, ... medientyp_n]"`

Optional: Das Attribut enthält eine kommagetrennte Liste aus einem oder mehreren Medientypen, für die die aktuelle Styleanweisung gelten soll.

Das group-Element

Das `<group>`-Element dient zum Gruppieren mehrerer XFrame-Definitionen, die per `<frame>`-Element vorgenommen werden. Das Element darf (optional und wiederholbar) eine beliebige Anzahl von `<group>`- oder `<frame>`-Elementen enthalten.

Anmerkung: Als Kindelement des Wurzelelements `<frames>` darf höchstens genau ein `<group>`-Element erscheinen.

```
<group compose="vertical">
    <frame/>
    <group compose="horizontal">
        <frame/>
        <frame/>
    </group>
    <frame/>
</group>
```

Attribute:

Es gilt der Standardattributsatz mit `xml:id`, `class` und `title`, sowie die elementspezifischen Attribute `compose` und `xml:base`.

```
compose = "vertical" (Default) | "horizontal" | "single" | "free" |
QName
```

Optional: Das `compose`-Attribut entscheidet über die Konfiguration der `<frame>`-Elemente der aktuellen Gruppe. Lässt man es weg, so wird, (falls die XFrame-Definitionen direkt als Kindelemente von `<frames>` auftreten, automatisch eine vertikale (Spalten-)Einteilung angenommen.

Hinweis: Eine mit `compose` definierte Konfiguration dient offiziell lediglich als »Vorschlag« und kann vom User Agent, falls erforderlich, anders interpretiert werden. Auch Styleanweisungen haben im Konfliktfall Vorrang. **[«]**

`compose = "vertical"`
Default: Ordnet die eingeschlossenen Frames oder Gruppen in Spalten an.

`compose = "horizontal"`
Ordnet die eingeschlossenen Frames oder Gruppen in Zeilen an.

`compose = "single"`
Zeigt genau einen der definierten Framebereiche. Dem Nutzer muss in Menüform die Anwahl der restlichen Framebereiche ermöglicht werden – die Methode ist nicht vorgeschrieben; für visuell orientierte User Agents ist die Realisierung mit Tabs oder vergleichbar jedoch impliziert.

`compose = "free"`
Die Framebereiche sind im Rahmen des verfügbaren Raumes beliebig positionierbar, dürfen sich daher auch überlappen und können vom User bewegt werden.

`compose = QName`
Bei jeder Implementierung von XFrames soll der betreffenden Anwendung die Möglichkeit einer eigenständigen Layout- oder Darstellungsvariante geboten werden. Hierfür ist für `compose` der Einsatz eines beliebig wählbaren `QName` als Wert gestattet. User Agents, die diesen Wert nicht interpretieren. sollen auf den Defaultwert `"vertical"` zurückfallen.

`xml:base = Basis-URI`

Optional: Einem `<group>`-Element kann mit Hilfe des `xml:base`-Attributs ein Basis-URI zugewiesen werden, der zum Auflösen relativer URIs der Ressourcen der enthaltenen Frames dient.

Das frame-Element

Das `<frame>`-Element dient der Definition eines XFrame-Bereiches. Es tritt entweder im Inneren eines `<group>`-Elements auf oder als Kindelement des Wurzelelements `<frames>`.

Attribute:

Es gilt der Standardattributsatz mit `xml:id`, `class` und `title` sowie das elementspezifische Attribute `source`.

`xml:id = Identifier`

Optional: In Zusammenhang mit dem `<frame>`-Element besitzt `xml:id` insofern eine besondere Bedeutung, als dass dem Bereich auf diese Weise sowohl die Styleanweisungen (Abmessung, Darstellung) als auch die darzustellende Dokumentressource zugewiesen wird.

[»] *Hinweis:* Hat ein Frame kein `xml:id`-Attribut, so kann ihm von außen keine Ressource zugewiesen werden. Besitzt er ein `source`-Attribut, so wird die dort definierte Ressource verwendet, ansonsten bleibt der Bereich leer.

`source = URI-Referenz`

Optional: Das `source`-Attribut definiert eine URI-Referenz auf das Default-Dokument des Framebereiches. Dieses Dokument wird automatisch dann dargestellt, wenn beim Aufruf des XFrames-Dokuments nicht von außen ein anderslautender Parameter übergeben wird – die Zuordnung der externen Ressourcendefinition zum Frame erfolgt über `xml:id`.

13.2.2 Beispiele für XFrames-Dokumente

Da es sich bei XFrames-Dokumenten um eine eigene XML-Anwendung und nicht um HTML- bzw. XHTML-Dokumente handelt, erhalten sie eine eigene Dateiendung: `.xfm` (auch die Endung `.xframes` wird vorgeschlagen). Ein XML-Prolog kann vorausgehen.

Eine Doctype-Deklaration würde wie folgt aussehen:

```
<!DOCTYPE frames PUBLIC "-//W3C//DTD XFrames 1.0//EN"
   SYSTEM "http://www.w3.org/MarkUp/DTD/xframes-1.dtd">
```

Das folgende XFrames-Dokument definiert einen dreizeiligen Frameset, dessen mittlerer Bereich in zwei Spalten unterteilt ist. Hierfür erhält die innere Gruppe den Wert `compose="horizontal"`. Für die äußere Gruppe, die kein `compose`-Attribut besitzt, gilt automatisch der implizite Wert `"vertical"`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE frames PUBLIC "-//W3C//DTD XFrames 1.0//EN"
   SYSTEM "http://www.w3.org/MarkUp/DTD/xframes-1.dtd">
<frames xmlns="http://www.w3.org/2002/06/xframes/">
  <head>
      <title>Dreizeiliges XFrames-Beispiel</title>
      <style type="text/css">
         #kopfbereich {height: 150px }
         #nav {width: 20%}
         #inhalt {width: 80% }
         #fussbereich {height: 50px }
      </style>
   </head>
   <group>
     <frame xml:id="kopfbereich"/>
     <!-- mittlerer Bereich: -->
     <group compose="horizontal">
        <frame xml:id="nav"/>
        <frame xml:id="inhalt"/>
     </group>
     <frame xml:id="fussbereich"/>
   </group>
</frames>
```

Listing 13.10 Dreizeilige XFrame-Konfiguration

Eine Konfiguration wie die folgende würde als »tabbed«-Frameset interpretiert. Hierfür erhält die (einzige) Gruppe das Attribut `compose="single"`:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE frames PUBLIC "-//W3C//DTD XFrames 1.0//EN"
   SYSTEM "http://www.w3.org/MarkUp/DTD/xframes-1.dtd">
<frames xmlns="http://www.w3.org/2002/06/xframes/">
   <head>
      <title>Tabbed XFrames-Beispiel</title>
   </head>
   <group compose="single">
      <frame xml:id="eins"/>
      <frame xml:id="zwei"/>
      <frame xml:id="drei"/>
   </group>
</frames>
```

Listing 13.11 Per Tab gesteuerte XFrame-Konfiguration

Ebenso ist eine Mischung aus verschiedenen Konfigurationen denkbar. Das folgende Beispiel erzeugt ein dreizeiliges Frameset mit einer per Tab gesteuerten Mittelgruppe und einem zusätzlichen, frei beweglichen Frame:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE frames PUBLIC "-//W3C//DTD XFrames 1.0//EN"
   SYSTEM "http://www.w3.org/MarkUp/DTD/xframes-1.dtd">
<frames xmlns="http://www.w3.org/2002/06/xframes/">
  <head>
      <title>Wilde Mischung</title>
      <style type="text/css">
         #kopfbereich { height: 150px }
         #fussbereich { height: 50px }
         #frei { position:absolute;
                 top:200px; left:300px;
                 width:200px, height:200px;
                 }
      </style>
   </head>
   <group>
     <frame xml:id="kopfbereich"/>
     <!-- drei tabbed Frames als mittlerer Bereich: -->
     <group compose="single">
        <frame xml:id="eins"/>
        <frame xml:id="zwei"/>
        <frame xml:id="drei"/>
     </group>
     <frame xml:id="fussbereich"/>
     <!-- ein verschiebbarer Bereich: -->
     <group compose="free">
        <frame xml:id="frei"/>
     </group>
   </group>
</frames>
```

Listing 13.12 Gemischtes XFrame-Dokument

13.2.3 Aufruf eines XFrames-Dokuments

Bei einem HTML-Frameset ist lediglich das Frameset-Dokument per URL aufrufbar, das dann seine Standardkonfiguration aufbaut. Ein Bookmark auf eine andere Konfiguration als die im Quelltext festlegte Ausgangskonfiguration ist nicht möglich. In XFrames ist das Problem dadurch gelöst, dass mit dem aufrufenden URL auch die XFrame-Konfiguration übergeben werden kann.

Zwar kann auch ein XFrame-Dokument analog zu einem HTML-Frameset in seiner Defaultkonfiguration aufgerufen werden, interessanter ist jedoch die Möglichkeit, mittels einer einem Fragment-Identifier ähnlichen Syntax `#frames()` die gewünschte Konfiguration zu spezifizieren.

Einem XFrame-Bereich kann so mit Hilfe seines vergebenen Identifiers ein Dokument unmittelbar aus dem URL zugewiesen werden. Der Aufruf

```
index.xfm#frames(eins=a.xhtml,zwei=b.xhtml,drei=c.xhtml)
```

weist z. B. in folgender Konfiguration den drei Framebereichen Dokumente zu:

```
...
<group>
      <frame xml:id="eins"/>
      <frame xml:id="zwei"/>
      <frame xml:id="drei"/>
</group>
...
```

Mittels eines solchen URL-Strings wird der Zugriff auf beliebige Frame-Konfigurationen möglich. Er kann obendrein als Bookmark abgelegt oder dynamisch erzeugt werden.

Da die Zuweisung über den Frame-Identifier erfolgt, ist die Übergabereihenfolge innerhalb des Strings irrelevant. Ein hierbei nicht berücksichtigter Framebereich erhält kein Dokument zugewiesen und lädt stattdessen das über sein `src`-Attribut festgelegte Default-Dokument. Eine Zuweisung, für die kein gleichnamiger Identifier im Frameset existiert, wird verworfen.

Zuweisung nur über ID

Eine Zuweisung eines Dokuments an einen XFramebereich ist per URI-String nur möglich, wenn der Frame einen Identifier besitzt.

13.3 Zusammenfassung

Aus diesem Kapitel sollten Sie folgende Aussagen mitnehmen:

- XForms und XFrames sind vielversprechende Technologien, die auf Zusammenarbeit mit XHTML 2.0 konzipiert sind, aber auch im XHTML-Kontext anwendbar sein werden.
- Während für XFrames aktuell noch keine Anwendungsmöglichkeiten besteht, existieren für XForms bereits client- und serverseitige Implementierungen.

- XForms verbssert herkömliche HTML-Formulare dahingegend, dass es Möglichkeiten der Validierung einschließt und einen Datenversand in XML-Form vorsieht.
- XFormular-Daten sind an ein Datenmodell gebunden. Die versandten Daten können anhand eines XML-Schemas reglementiert und validiert werden.
- Der Datenversand eines XFormulars kann parallel an mehrere Adressaten erfolgen (mehrfaches Submit).
- Es stehen erweiterte Eingabemöglichkeiten zur Verfügung, darunter Datums- und Bereichsfelder.
- XFrames ist eine XML-Anwendung, die die Frames-Funktionalität von HTML in XFrames-Dokumenten nachbildet.
- XFrames beseitigen bestehende Mängel von HTML-Frames. Sie sind vollständig bookmarkbar und bieten weitgehende Freiheit im Framelayout.

Wie geht es weiter?

Das folgende Kapitel 14, *(X)HTML Elementreferenz*, bietet Ihnen eine komplette Übersicht über alle existierenden HTML-Elemente, ihre Anwendung, ihrer Attribute und deren Werte. Es folgt mit Kapitel 15, *(X)HTML-Attributreferenz* ein vertiefender Überblick über die Universalattribute und ihre Anwendung.

In diesem Kapitel finden Sie eine vollständige Referenz zur Verwendung aller existierenden Elemente aus HTML und XHTML mit Auflistung und Erläuterung ihrer sämtlichen Attribute und deren jeweils erlaubten Werte.

14 (X)HTML-Elementreferenz

Da dieses Kapitel den umfangreichsten und (aus meiner Sicht) zentralen Teil dieses Werks darstellt, rechtfertigt sich hier ein eigenes, kleines Vorwort. Dargestellt sind auf den nächsten, knapp über 500 Seiten sämtliche vom W3C spezifizierten Elemente aus HTML sowie zwei nicht spezifizierte, die in der Praxis dennoch oft benötigt werden und daher im Einsatz sind (nämlich `<marquee>` und `<embed>`). Auch ältere, offiziell spezifizierte, aber inzwischen abgeschaffte Elemente (es sind nur wenige) sind in knapper Form aufgenommen – der Platzbedarf war zu gering, um ihr Weglassen zu rechtfertigen, zumal auch sie in aktuellen Browsern weiterhin unterstützt werden.

Bei den Attributen der Elemente ging es mir ebenfalls um Vollständigkeit, weshalb jeweils auch als »deprecated« vorgesehene, aber durchaus noch gültige Präsentationsattribute sowie proprietär implementierte Attribute aufgenommen wurden. Letztere wurden gerade durch Microsoft für den Internet Explorer in reicher Auswahl eingeführt. Sofern Sie (beispielsweise innerhalb von Intranets) für diesen Browser als Zielplattform entwickeln, sollten Sie über diese erweiterten Features informiert sein. Das Gleiche gilt für andere, aus europäischer Sicht exotischer anmutende Plattformen wie DoCoMo iMode oder WebTV.

Beachten Sie jedoch, dass der Einsatz proprietärer Elemente oder Attribute eine erfolgreiche Validierung Ihres Dokuments gegenüber den offiziellen Spezifikationen verhindert.

Diejenigen Attribute, die als Universalattribute (beinahe) allen Elementen zugeordnet sind, werden in diesem Zusammenhang nur dort erwähnt, wo sie für das entsprechende Element von zentraler Bedeutung sind oder ihre Verwendung von der gewöhnlichen abweicht. Ansonsten sind entsprechende Erläuterungen wie auch die der Eventhandler-Attribute in das folgende Kapitel 15*, (X)HTML-Attributreferenz,* ausgelagert.

Als Grundlage der folgenden Beschreibungen dienten mir die – leider teilweise widersprüchlichen – W3C-Empfehlungen zu HTML und XHTML und dort beschriebene DTDs sowie Unterlagen der Browserhersteller. Wo die Praxislage unklar war (was ziemlich häufig war), stellte ich eigene Experimente an. Hierbei habe ich mich auf aktuelle Browser beschränkt (Internet Explorer 5 bis 7, Mozilla ab 1.5, Firefox ab 1.0, Opera ab 7.0).

Alphabetische Übersicht der Elemente

a, S. 411
abbr, S. 430
acronym, S. 433
address, S. 437
applet, S. 439
area, S. 446
b, S. 454
base, S. 458
basefont, S. 463
bdo, S. 466
big, S. 473
blockquote, S. 476
body, S. 481
br, S. 493
button, S. 498
caption, S. 503
center, S. 507
cite, S. 510
code, S. 512
col, S. 515
colgroup, S. 521
dd, S. 527
del, S. 530
dfn, S. 533
dir, S. 536
div, S. 539
dl, S. 549
dt, S. 554
em, S. 557
embed, S. 560
fieldset, S. 569
font, S. 573
form, S. 580
frame, S. 592
frameset, S. 604
h1 – h2, S. 615
head, S. 618
hr, S. 621
html, S. 626
i, S. 633
iframe, S. 635
img, S. 643
input, S. 655
ins, S. 671
isindex, S. 674
kbd, S. 679
label, S. 681
legend, S. 685
li, S. 688
link, S. 693
listing, S. 704
map, S. 706
marquee, S. 708
menu, S. 715
meta, S. 719
nextid, S. 738
noframes, S. 740
noscript, S. 744
object, S. 748
ol, S. 759
optgroup, S. 765
option, S. 768
p, S. 772
param, S. 775
plaintext, S. 779
pre, S. 781
q, S. 785
s, S. 789
samp, S. 791
script, S. 794
select, S. 803
small, S. 809
span, S. 812
strike, S. 815
strong, S. 817
style, S. 819
sub, S. 826
sup, S. 828
table, S. 830
tbody, S. 845
td, S. 851
textarea, S. 863
tfoot, S. 871
th, S. 875
thead, S. 887
title, S. 892
tr, S. 894
tt, S. 900
u, S. 903
ul, S. 905
var, S. 910
xmp, S. 912

14.1 a

Das Element <a> (*anchor*) definiert wahlweise einen Ankerpunkt (Linkziel) oder den Ausgangspunkt eines Hyperlinks.

14.1.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Hypertext
Kontext	Gestattet in allen Block- oder Inlineelementen, außer in <a> selbst
Erlaubte Inhalte	Text- und Inlinelementinhalt außer <a> selbst
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, iMode 3.0, 4.0, HTML 4.0 mobile iMode XHTML1.0, 1.1; XHTML mobile profile

Syntaxbeispiel:

```
<a href="beispiel.html"> ... </a>
<a name="ankerpunkt"></a>
<a href="beispiel.html" name="ankerpunkt"> ... </a>
```

Defaultrendering:
Mit definierte Ankerpunkte sind unsichtbare Marker. Der Textinhalt eines Hyperlinks ... wird in Defaultlinkfarbe (blau bzw. lila für besuchte, rot für aktive Links) mit Unterstrich dargestellt, enthaltene Grafiken mit Rahmen in Default-Linkfarbe. Ein im Fokus befindlicher Link erhält eine gepunktete Umrisslinie (*outline*).

CSS-Umsetzung des HTML-Defaults:

```
a, a[name]        { display: inline }
a[href]            { text-decoration: underline;
                     color: #0000ff; }     /* blau */
a[href]:visited    { color: #bb77ff }      /* lila */
a[href]:active     { color: #ff0000 }      /* rot  */
a[href]:focus      { outline: thin dotted invert; }
```

14.1.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	`onblur`, `onfocus`
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onerrorupdate`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`accesskey`, `charset`, `coords`, `href`, `hreflang`, `methods`*, `name`, `rel`, `rev`, `shape`, `tabindex`, `target`, `title`, `type`, `urn`* *abgeschafft
Internet Explorer	`disabled`, `hidefocus` *Access-, Databinding-, Edit-Attribute*
Netscape	`suppress`
DoCoMo iMode	`cti`, `email`, `ijam`, `ilet`, `irst`, `ista`, `iswf`, `kana`, `telbook`, `utn`
Web Application 1.0	`ping`

14.1.3 Offizielle Attribute – W3C

accesskey

Wert	*[a-z][1–0]*#*
Typ	Alphanumerisches Zeichen sowie * und #; nicht *case-sensitive*.
Verwendung	Optional

Barrierefreiheit: Das Attribut `accesskey` bestimmt im Sinne der Accessibility ein einzelnes alphanumerisches Tastaturzeichen, das im Sinne eines Keyboard-Short-

cuts den Link ausführt – unter Windows bei gleichzeitig gedrückter Alt-Taste[1], unter Mac OS mit Ctrl-Taste.

```
<a href="beispiel.html" accesskey="c"
 onfocus="alert('Link aktiviert!')">
 Alt + C eingeben um Link auszuwählen</a>
```

Der Link im oberen Beispiel wird mit Alt+C ausgewählt. (Im Internet Explorer ist Bestätigen der Linkausführung per ↵-Taste erforderlich.)

Abbildung 14.1 Link über Accesskey aktiviert

Hinweis: Auf Webseiten für mobile Endgeräte sollten als Accesskeys lediglich die Ziffern 0–9 verwendet werden. Des Weiteren sind auch * (Asterisk) und # (Hash-Symbol) gestattet, wobei (geräteabhängig!) die durch die entsprechenden Tasten gesteuerte Seitenscrollfunktion Vorrang vor den Accesskeys besitzt. **[«]**

charset

Wert	*encoding-bezeichner*
Typ	Ein registrierter Bezeichner für Character-Encodings
Verwendung	Optional

Das charset-Attribut teilt dem Browser das zu erwartende Encoding-Schema des Linkziels mit. Default-Encoding ist ISO 8859–1 (ISO Latin 1), eine Untergruppe von ISO 8859 (siehe **www.iana.org/assignments/character-sets**).

```
<a href="beispiel.html" charset="ISO-8859-8">
 Das Zieldokument ist in ISOLatinHebrew abgefasst und
 verwendet hebräische Schriftzeichen</a>
```

1 Unter Windows in Opera-Browsern statt mit Alt mit gleichzeitig gedrückter ⇧- und Esc-Taste. Hintergedanke ist, den Tastaturzugang zum Browsermenü per Alt-Taste nicht zu behindern. Dieser bleibt jedoch bei einzelner Betätigung der Alt-Taste auch bei den anderen Browsern unbeschränkt möglich.

coords

Wert	*x_koordinate_1, y_koordinate_1 [, x_koordinate_2, y_koordinate_2 ...]*
Typ	kommagetrennte Zahlenwerte, als Pixelwert interpretiert
Verwendung	Optional, aber gemeinsam mit `shape`-Attribut

Die Wirkung des `coords`-Attribut bei `<a>` entspricht dem gleichnamigen Attribut bei `<area>`.

[!] *Achtung:* Das `coords`-Attribut wird in Zusammenhang mit `<a>` derzeit in keinem Browser unterstützt.

href

Wert	*URI-String*
Typ	Absoluter oder relativer URL der verlinkten Ressource
Verwendung	Optional. Wenn kein `name`-Attribut vorhanden ist, ist das `href`-Attribut **obligatorisch**.

Das `href`-Attribut nennt der URL der Ressource, auf die der Link zielt. Am Anfang des URI-Strings kann mit unmittelbar folgendem Doppelpunkt ein Bezeichner eines zu verwendenden Protokolls oder Pseudoprotokolls stehen. Dies ermöglicht beispielsweise per `mailto`-Link das direkte Ansprechen des E-Mail-Clients oder als `javascript`-Link des Scriptinterpreters des Browsers. Das Default-Präfix ist `http`.

Beispiel für einen JavaScript-Link:

```
<a href="javascript:alert('Beispiel!')">
 Beispiel für einen JavaScript-Link</a>
```

Beispiel für einen Mailto-Link:

```
<a href="mailto:ich@beispieldomain.de">Mail an mich</a>
```

Mehr oder weniger allgemein unterstützt sind die **Präfixe**[2] `about`, `ftp`, `http`, `mailto`, `javascript`, `news` und gegebenenfalls auch `telnet`.

Internet Explorer: Im Internet Explorer wird auch das Präfix `callto` unterstützt, das Online-Konferenzen mit Microsoft NetMeeting ermöglicht.

Do Co Mo: Im Mobile Web ist auch das Präfix `tel` verbreitet, das den URL als anzuwählende Telefonnummer deklariert.

2 Eine komplette Liste der einschlägigen und auch einiger exotischer Linkpräfixe hält das W3C bereit: www.w3.org/Addressing/schemes.html.

```
<!-- nicht existente Beispielnummer -->
<a href="tel:+493012345678">030-12345678</a>
```

Eine Telefonnummer darf maximal 24 Zeichen lang sein. Es wird empfohlen, stets die internationale Vorwahl anzugeben.

hreflang

Wert	*lang-bezeichner*
Typ	Ein Sprachenbezeichner nach RFC 1766
Verwendung	Optional
Einführung	HTML 4.0

Das Attribut `hreflang` nennt die Sprache, in der die über den Link adressierte Ressource abgefasst ist (RFC 1766: siehe **www.ietf.org/rfc/rfc1766.txt**).

```
<a href="paris.html" hreflang="fr">Info über Paris
 (in französischer Sprache </a>
```

methods

Wert	*http-methods*
Typ	HTTP-Methode zur Linkansprache
Verwendung	Optional
Einführung	HTML 2.0 **abgeschafft** seit HTML 3.2

(Historisch) Das `methods`-Attribut sollte eine HTTP-Methode nennen (oder als leerzeichengetrennte Liste auch mehrere von ihnen), mittels der das Linkziel vorzugsweise angesprochen werden sollte. Als Werte waren beispielsweise `"text-search"`, `"get"`, `"head"` und weitere vorgesehen.

Das Attribut wird abgesehen vom Internet Explorer (ab 4.0) nicht unterstützt und ist bereits seit HTML 3.2 nicht mehr Teil der Spezifikation.

name

Wert	*nmtoken*
Typ	Nametoken als Ankerpunktbezeichner
Verwendung	Optional
Einführung	HTML 2.0 **abgeschafft** in XHTML 1.1 zugunsten von `id`

Enthält ein NAMETOKEN, der das Element zum adressierbaren Ankerpunkt macht. Ein NAMETOKEN darf aus beliebigen alphanumerischen Zeichen, Punkt, Binde- und Unterstrich sowie Doppelpunkt bestehen (keine Leerzeichen).[3]

[»] *Hinweis:* Wenn kein href-Attribut vorhanden ist, ist das name-Attribut **obligatorisch**. Ein reiner name-Anker ist unsichtbar und braucht keinen Inhalt zu besitzen. Ein Inhalt ist jedoch gestattet: Enthält ein Ankerpunktcontainer einen längeren Textabschnitt, so zielt ein auf den Ankerpunkt gerichteter Link stets auf den Beginn des Bereichs. Die Endmarke ist in jedem Fall obligatorisch (auch eine Verkürzung wie <a name="mein_anker"/> ist nicht gestattet).

```
<p><a name="absatz1"></a>
 Der Ankerpunkt am Beginn dieses Absatzes kann
 über einen Link angesprochen werden.</p>
```

Das name-Attribut übernimmt in diesem Zusammenhang eine ähnliche Rolle wie ein id-Attribut, wobei ein id-Identifier dokumentweit eindeutig sein muss, also kein zweites Mal (auch bei einem anderen Elementtyp) erscheinen darf.

Werden name und id gleichzeitig verwendet (was bis einschließlich XHTML 1.0 gestattet ist), müssen beide Attribute den gleichen Wert erhalten:

```
<p><a name="absatz1" id="absatz1"></a>
 Der Ankerpunkt am Beginn dieses Absatzes kann
 über einen Link angesprochen werden.</p>
```

Als gleichwertiger Ersatz kann auch ein ID im Absatz verwendet werden:

```
<p id="absatz1">Dieser Absatz kann dank seines Identifiers
 ebenfalls direkt über einen Link angesprochen werden.</p>
```

Von aktuellen Browsern wird ein Element mit id-Identifier ebenfalls als Ankerpunkt akzeptiert, so dass der Ankerpunkt mit <a name="..."> im Wesentlichen der Abwärtskompatibilität zu älterenBrowsern dient.

Groß- und Kleinschreibung von id- und name-Bezeichnern

Die Werte von id- und name-Attributen sind nicht zuverlässig *case-sensitive*, obwohl sie es laut Spezifikation sein sollten.

Aus historischen Gründen (wenn auch nicht mehr gängige Praxis) ist es dem Browser gestattet, ein so genanntes »case folding« hin zu Kleinbuchstaben vorzunehmen – es ist daher möglich, dass name="x" und name="X" nicht unterschieden werden. Die Spezifikation *untersagt* daher innerhalb eines Dokuments solche Wertdoubletten für name, die sich nur durch die Schreibweise unterscheiden.

3 Ein ID besitzt zusätzlich die Einschränkung, dass er nicht mit einer Ziffer beginnen darf (also ein so genannter *XML-name* sein muss). Seit HTML 4.01 unterliegt auch der Wert des name-Attributs dieser Beschränkung.

rel

Wert	*rel-bezeichner*
Typ	Vordefinierter Bezeichner für Link-Relationen.
Verwendung	Optional

Ein vordefinerter Relation-Bezeichner oder eine durch Leerzeichen getrennte Liste mehrerer dieser Bezeichner. Das `rel`-Attribut beschreibt die Beziehung (*relation*) der verlinkten Ressource zum verlinkenden Dokument.

Vordefinierte Werte sind: `alternate`, `appendix`, `bookmark`, `chapter`, `contents`, `copyright`, `designates`, `glossary`, `help`, `index`, `next`, `nofollow`, `prev`, `section`, `start`, `stylesheet`, `subsection`.

Aus dem Rahmen fällt der Wert `rel="nofollow"`, der Suchmaschinen untersagt, dem Verweis zu folgen (als Alternative zur pauschalen Festlegung für das gesamte Dokument per `<meta>`-Tag). Der Wert wird von Google als Lösung des »Blog Noise«-Problems propagiert und ist in verschiedener Bloggingsoftware bereits »Standarddreingabe« zu Kommentarlinks. Das Schlüsselwort beschreibt jedoch eher eine »Nichtbeziehung« als eine Beziehung.

Tipp: Das rel-Attribut in Verbindung mit Dublin Core

Auch Beschreibungen (»Elemente«) nach **Dublin Core** eigenen sich als Wert des `rel`-Attributs. DC-Elemente können beispielsweise in Intranet- oder Bibliotheksanwendungen eingesetzt werden, wobei beispielsweise das Element `DC.relation` mit den entsprechenden Refinements die Beziehungen des Quelldokuments zur verlinkten Ressource exakt definieren kann – in folgendem Beispiel als Quelle, auf die im aktuellen Dokument Bezug genommen wird.

```
<a href="manual.htm" rel="DC.relation.references">Manual</a>
```

Hinweis: Eine Übersicht aller DC-Elemente und ihrer Bedeutung erhalten Sie im Abschnitt zum `<meta>`-Element (siehe Abschnitt 14.55). **[«]**

rev

Wert	*rel-Bezeichner*
Typ	Vordefinierter Bezeichner für Link-Relationen.
Verwendung	Optional

Enthält einen vordefinierten Relation-Bezeichner oder eine durch Leerzeichen getrennte Liste aus mehreren dieser Bezeichner.

Das `rev`-Attribut dient in Umkehrung zum `rel`-Attribut dazu, die Beziehung des verlinkenden Dokument zur verlinkten Ressource zu beschreiben. Als vordefinierte Werte gelten (mit der gleichen Einschränkung) ebenfalls die für das `rel`-Attribut angegebenen Werte.

Anmerkung: Eine Anwendung der Dublin Core Elemente ist auch für dieses Attribut denkbar.

shape

Wert	*"rect"* | *"circle"* | *"poly"* | *"default"*
Typ	Bezeichner für die Form eines anklickbaren Bereichs
Verwendung	Optional, aber gemeinsam mit `coords`-Attribut

Die Wirkung des `shape`-Attribut bei `<a>` entspricht dem gleichnamigen Attribut bei `<area>`.

Als browserspezifische Synonyme der offiziellen Werte werden gelegentlich folgende Werte eingesetzt `"rectangle" | "circ" | "polygon"`.

[!] *Achtung:* Das `shape`-Attribut wird in Zusammenhang mit `<a>` derzeit in keinem Browser unterstützt.

tabindex

Wert	*number*
Typ	Eine Ganzzahl
Verwendung	Optional

Barrierefreiheit: Die im `tabindex`-Attribut übergebene Ganzzahl legt in aufsteigender Richtung die Position des Links in der Tabulatorwahl-Reihenfolge (⇆-Taste) fest.

Die Tab-Reihenfolge gestaltet sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes `tabindex`-Attribut in Quelltextreihenfolge. (Zu den aktivierbaren Elementen gehören allgemein neben `<a>` auch `<area>`, `<button>`, `<input>`, `<object>`, `<select>` und `<textarea>`.)

Anmerkung: Ein negativer Tabindex-Wert entfernt den Link im IE vollständig aus der Tab-Reihenfolge. Die anderen Browser ordnen ihn in die normale Tab-Reihenfolge ein, als ob kein `tabindex`-Attribut gesetzt wäre.

target

Wert	*nmtoken*
Typ	Nametoken als Frame- oder Fensterbezeichner
Verwendung	Optional
Gültigkeit	nicht in XHTML 1.0 strict

Ein freigewählter Bezeichner eines (existierenden) Browserfensterobjekts oder im Framesetkontext einer der vordefinierten Bezeichner `"_self"`, `"_parent"`, `"_top"` oder `"_blank"`. Ein explizit im Link gesetztes Target hat Vorrang vor einem mit Hilfe des `<base>`-Elements und dessen `target`-Attribut gesetzten globalen Targets.

Achtung: Das `target`-Attribut ist in den *strict*-DTDs nicht enthalten. **[!]**

Der Hyperlinkbefehl wird in dem über das `target`-Attribut bezeichneten Fenster oder Frame ausgeführt:

- `target="_self"`
  Der Link wird im gleichen Fensterobjekt des aktuellen Framekontexts ausgeführt.
- `target="_parent"`
  Der Link wird im unmittelbar dem aktuellen Frame übergeordneten Frameobjekt des aktuellen Framekontexts ausgeführt.
- `target="_top"`
  Der Link wird im hierarchisch obersten Frame des aktuellen Framekontexts ausgeführt (auch als »Hauptfenster« bezeichnet; der Frameset wird entladen).
- `target="_blank"`
  Der Link wird in einem, sich außerhalb des aktuellen Framekontexts neu öffnenden, Fenster ausgeführt.
- `target="[nametoken]"`
  Ein frei gewählter Bezeichner in Form eines `NMToken` führt den Link im gleichnamigen Fensterobjekt aus (falls vorhanden – dies kann ein benanntes Frameobjekt oder ein frei schwebendes Fenster sein) oder öffnet ein Browserfenster, das anschließend unter diesem Bezeichner ansprechbar ist.

[»] *Hinweis:* Der vordefinierte Bezeichner `"_blank"` öffnet ebenfalls ein neues Fenster, vergibt dabei aber keinen Bezeichner, so dass dieses Fenster in Folge über Links *nicht* erneut ansprechbar ist.

Besonderheit in modularem XHTML
Die Verwendung des `target`-Attributs erfordert die Einbindung des XHTML-Moduls »Target«. Das Attribut ist in XHTML 1.0 »*strict*« nicht enthalten.

title

Wert	*cdata*
Typ	Beliebiger Textinhalt
Verwendung	Optional

Das `title`-Attribut dient dazu, einem Link einen erklärenden Text beizugeben, der bei Mouseover als Tooltip gezeigt werden kann. Das Attribut gehört zu den Standardattributen, hat aber in Verbindung mit dem `<a>`-Element besondere Wichtigkeit.

Anmerkung: Im Internet Explorer wird der Wert des `title`-Attributs als Label der Favoritenliste verwendet, wenn ein Link dieser über das Kontextmenü hinzugefügt wird.

type

Wert	*cdata*
Typ	Ein MIME-Typ-Bezeichner
Verwendung	Optional

Das Attribut `type` nennt den MIME-Typ (Multipurpose Internet Mail Extension) der externen Ressource, auf die der Link zielt. Die Angabe besteht aus der Angabe eines Hauptmedientyps und einem Subtyps, die durch Slash getrennt sind – z. B.: `"text/html"`, `"text/css"`, `"image/gif"`, `"application/xml"` sowie eine Reihe weiterer (siehe RFC 2045, 2046 und 2077).

urn

Wert	*cdata*
Typ	Beliebiger Textinhalt in Form eines URN
Verwendung	Optional
Einführung	HTML 1.0 **abgeschafft** seit HTML 3.2

(Historisch) Das `urn`-Attribut war als Alternative zum `name`-Attribut gedacht und sollte die Verwendung von URNs (Universal Resource Names) für Links ermöglichen.

14.1.4 Proprietäre Attribute – Internet Explorer

disabled (IE5)

Wert	*"disabled"*
Typ	Festgelegter Bezeichner
Verwendung	Optional

Das `disabled`-Attribut kennzeichnet den Link durch Ausgrauung als inaktiv und nimmt ihn gleichzeitig aus der Tab-Reihenfolge.

Ein eventuell gesetztes `accesskey`-Attribut wird ebenfalls deaktiviert. Ein Ansprechen des Links über die Tastatur ist somit nicht mehr möglich.

Achtung: Der Link wird durch Setzen des Attributs auch im Internet Explorer nicht wirklich deaktiviert, sondern bleibt per Maus anklickbar und funktionstüchtig! **[!]**

hidefocus (IE5.5)

Wert	*"true", "false"*
Typ	Festgelegter Bezeichner
Verwendung	Optional

Das `hidefocus`-Attribut verhindert im Internet Explorer das Erscheinen der gepunkteten Focus-Outline beim Aktivieren des Links.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (ab IE 5.5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"` (IE 5)

`unselectable = "on" | "off"` (IE 5)

14.1.5 Proprietäre Attribute – Netscape

suppress (NN4)

Wert	*"false"* | *"true"*
Typ	Boolesche Werte
Verwendung	Optional

(Historisch) Das `suppress`-Attribut unterdrückt die gepunktete Focus-Outline, die ansonsten um einen per Tastatur ausgewählten Link erscheint. Das Attribut wird in aktuellen Mozilla- oder Netscape-Browsern nicht mehr unterstützt.

14.1.6 Proprietäre Attribute – DoCoMo iMode

cti

Wert	*0, 1, 2, ... 9 # * , /*
Typ	beliebige Folge aus der Gruppe festgelegter Zeichen
Verwendung	Optional
Einführung	ab iMode 2.0, cHTML 2.0

Tonwahl: Das Attribut `cti` übergibt eine Zeichenfolge, die als tonwahlgeeignete Telefonnummer mit optionalen zusätzlichen Steuerbefehlen (Pause, Eingabeaufforderung) verwendet wird.

Erlaubte Zeichen sind die Ziffern `0` bis `9`, das Komma (d.i.: »1 Sekunde warten«; wiederholbar), der Slash (d.i.: »auf Tastastureingabe warten«) sowie Stern `*` und Raute `#`. Die Angabe des `tel`-Protokolls ist im `cti`-Attribut im Gegensatz zum `href`-Attribut, wenn dieses eine Telefonnummer aufnehmen soll, nicht erforderlich. Bei gleichzeitig vorhandenem `href`-Attribut hat das `cti`-Attribut in einer iMode-Anwendung Vorrang.

Das folgende Beispiel wählt zunächst die (fiktive) Nummer `+49301234567`:

```
<a href="tel:+49301234567"
   cti="+49301234567/,,12345#,678#">030-1234567</a>
```

Anschließend wird, sobald die Gegenseite das Gespräch annimmt, eine Tastatureingabe erwartet (-> Slash). Nachdem diese erfolgt ist, wird nach einer Wartezeit von zwei Sekunden (-> zwei Kommas) die Ziffernfolge `#12345` gewählt, erneut eine Sekunde gewartet und abschließend `#678` gewählt.

Geräte, die das `cti`-Attribut nicht unterstützen, verwenden stattdessen die Nummer des `href`-Attributs.

email

Wert	*cdata*
Typ	Zeichenkette
Verwendung	Optional
Einführung	i-mode HTML 4.0 (FOMA-Extension)

Telefonbuch: Das `email`-Attribut erleichtert das Übernehmen einer E-Mail-Adresse ins interne Telefonbuch des Endgeräts. Maximale Länge 50 Zeichen. Das Attribut kann jedoch nicht in einem Link verwendet werden, der gleichzeitig über `href="mailto:..."` eine E-Mail-Adresse anspricht (das `tel`-Präfix wie im Beispiel funktioniert hingegen).

```
<p>Hans Mustermann zum <a href="tel:+49301234567" telbook="Hans
Mustermann" email="hans@mustermann.com">Telefonbuch</a>
hinzuf&uuml;gen</p>
```

ijam

Wert	*idref*
Typ	Boolesche Werte
Verwendung	Optional
Einführung	i-mode HTML 3.0

iopli-Anwendungen: Das `ijam`-Attribut referenziert ein `<object>`-Element, das eine javabasierte ippli-Anwendung geladen hat. Darf nicht gleichzeitig mit dem `ista`-Attribut verwendet werden.

ilet

Wert	*idref*
Typ	Idreferenz auf ein <object>-Element
Verwendung	Optional
Einführung	i-mode HTML 5.0

iopli-Anwendungen: Das `ilet`-Attribut referenziert ein `<object>`-Element, das eine javabasierte iopli-Anwendung geladen hat. Gleichzeitig muss das `href`-Attribut vorhanden sein, um Geräten, die keine iopli-Anwendungen unterstützen, eine alternative Zielseite anzubieten.

```
<a ilet="#object-id" href="alternativURL"> ... </a>
```

irst

Wert	*idref*
Typ	Idreferenz auf ein <object>-Element
Verwendung	Optional
Einführung	i-mode HTML 5.0

Infrarotschnittstelle: Das `irst`-Attribut referenziert ein `<object>`-Element, das die per Infrarotschnittstelle empfangenen Daten entgegennehmen soll.

ista

Wert	*idref*
Typ	Idreferenz auf ein <object>-Element
Verwendung	Optional
Einführung	i-mode HTML 4.0

Java-Anwendungen: Das `ista`-Attribut referenziert ein `<object>`-Element, das eine bereits geladene javabasierte Java-Anwendung enthält. Gleichzeitig muss das `href`-Attribut vorhanden sein, um Geräten, die keine Java-Anwendungen unterstützen, eine alternative Zielseite anzubieten.

```
<a ista="#object-id" href="alternativURL"> ... </a>
```

iswf

Wert	*idref*
Typ	Idreferenz auf ein `<object>`-Element
Verwendung	Optional
Einführung	i-mode HTML 5.0

Flash-Anwendungen: Mit dem `iswf`-Attribut kann ein `<object>`-Element angesprochen werden, das einen Flash-Film lädt, sofern das Gerät in interaktiven Modus geschaltet ist. Für Geräte, die nicht in interaktiven Modus geschaltet sind oder diesen nicht unterstützen, zeigt das `href`-Attribut gleichzeitig auf eine alternative Ressource, die keine Flash-Funktionalität benötigt.

```
<a iswf="#object_tag_id" href="URL_alternativinhalt">
   ... </a>
```

Das angesprochene `<object>`-Element benötigt ein `id`-Attribut, dessen Wert dem hier verwendeten `IDREF` entspricht, sowie ein gesetztes `declare`-Attribut, das den Tag als Flashobjekt kenntlich macht.

kana

Wert	*cdata*
Typ	Zeichenkette
Verwendung	Optional
Einführung	i-mode HTML 4.0 (FOMA-Extension)

Telefonbuch: Das `kana`-Attribut kann alternativ oder zusätzlich zum `telbook`-Attribut einen Namenseintrag in japanischen, so genannten »half-size« Kana-Schriftzeichen (Silbenschriftzeichen) ermöglichen. Die maximale Länge beträgt 18 Zeichen.

Anmerkung: Japanische Mobiltelefone besitzen auch eine Tastaturbelegung mit Hiragana-Schriftzeichen. Westliche Schriftzeichen werden in diesem Zusammenhang als »Romaji« bezeichnet.

telbook

Wert	*cdata*
Typ	Zeichenkette
Verwendung	Optional
Einführung	i-mode HTML 4.0 (FOMA-Extension)

Telefonbuch: Das `telbook`-Attribut ermöglicht, den Namen des über den Link adressierten Teilnehmers dem internen Telefonbuch hinzuzufügen. Der Eintrag darf maximal 20 Zeichen (je 8 Bit, also 20 Bytes) lang sein.

```
<p>Hans Mustermann zum
<a href="tel:+49301234567" telbook="Hans Mustermann"
email="hans@mustermann.com">Telefonbuch</a>
hinzuf&uuml;gen</p>
```

utn

Wert	*"utn"*
Typ	festgelegte Werte
Verwendung	Optional
Gültigkeit	iMode 3,0, 4.0; iMode XHTML 1.0, 1.1

Identifizierung: Durch Setzen des `utn`-Attributs wird vom Server eine persönliche Teilnehmeridentifizierung abgefragt, also beispielsweise das Versenden der Seriennummer des Endgeräts bzw. zusätzlich die von seiner FOMA-Card[4] (iMode). Das Attribut wird nicht von allen Endgeräten unterstützt. Der Teilnehmer kann das Versenden der Kennung verweigern.

14.1.7 Proprietäre Attribute – Web Application 1.0

ping

Wert	*"URI_a [URI_b URI_c … URI_n]"*
Typ	ein oder mehrere URI
Verwendung	Optional

User-Tracking: Eine leerzeichengetrennte Liste aus einem oder mehreren URIs, an die bei Klick auf den Link durch den Browser ein Ping-Befehl geschickt wird. Der Link soll dem User als Tracking-Link kenntlich gemacht werden und das Ping-Verhalten soll deaktivierbar sein.

Das Attribut stellt eine von der WHAT WG[5] vorgeschlagene Erweiterung dar und ist nicht unumstritten. Es wird von **Firefox** ab Version 1.6 unterstützt.

14.1.8 Erläuterungen zum Element

Der wichtigste Aspekt an HTML ist die Möglichkeit, Dokumente durch »Hyperlinks« miteinander zu verknüpfen. Ohne Hyperlinks gäbe es kein Internet, sondern nur eine Menge isolierter Dokumente. Hyperlinks stellen eine Abkürzung zu einer Information dar, die an irgendeiner anderen Stelle liegen kann – innerhalb desselben Dokuments oder aber auch auf einem ganz anderen Server. Der `<a>`-Container kann daher mit Recht als der zentrale Tag von HTML bezeichnet werden.

Ein Hyperlink wird durch den so genannten »Anker«-Container definiert; eine Endmarke `</a>` ist obligatorisch. Pflichtattribute existieren nicht. Ohne Attribute kommt das Element in der Praxis jedoch nicht vor – vorgeschrieben sind mindestens das `href`-Attribut *oder* das `name`- bzw. `id`-Attribut. Im Zusammenhang mit JavaScript werden innerhalb des Start-Tags des Ankers auch Eventhandler-Attribute, meist für Mausereignisse, benötigt.

4 FOMA ist ein DoCoMo-Akronym für »Freedom of Mobile Multimedia Access«.

5 »Web Hypertext Application Technology Working Group«, eine Initiative verschiedener Browserhersteller, die abseits vom W3C an einer Weiterentwicklung von HTML arbeiten – siehe http://whatwg.org.

Erlaubte Inhalte

Als Inhalte von `<a>` sind in den *strict*-DTDs Text (`#PCDATA`) und sämtliche Inlineelemente zur Textformatierung gestattet sowie `<map>`, `<object>` und `<script>`, außerdem die Formularinputelemente (nicht: `<fieldset>`). Die *transitional*-DTDs erlauben als Inhalt zusätzlich `<applet>` und `<iframe>`.

Als Inhalt *ausgeschlossen* ist `<a>` selbst. *Nicht gestattet* sind des Weiteren alle Blockelemente sowie `<noscript>` und `<noframes>`.

Pfadangaben in URL-Strings

Für das korrekte Auffinden des Zieldokuments ist beim Definieren des Hyperlinks eine Pfadangabe erforderlich. Man unterscheidet zwischen relativen Pfadangaben, die ausgehend vom Ausgangsdokument definiert sind, sowie wurzelbezogenen und absoluten Pfadangaben, die nicht von der Position des Ausgangsdokuments abhängen.

- Eine **relative Pfadangabe**
  wird für siteinterne Navigation verwendet.

  Die relative Pfadangabe hat den Nachteil, dass der Bezug zwischen dergestalt verlinkten Dokumenten verloren geht, sobald das Ausgangsdokument verschoben wird (z. B. in ein anderes Verzeichnis – die Pfadangabe führt in diesem Falle ins Leere).

  Beispiel für eine relative Pfadangabe:

  ```
  <a href="meine_seite.html">Meine andere Seite</a>
  ```

  Eine relative Pfadangabe wird zum **Basis-URI** des aufrufenden Dokuments aufgelöst. Der Basis-URI entspricht dessen realer Webadresse; er kann jedoch auch über den `<base>`-Tag (s. dort) explizit gesetzt werden.

- Eine **absolute Pfadangabe**
  wird für siteübergreifende bzw. absolute Adressierung benötigt.

  Die absolute Pfadangabe ist gegenüber einer Verschiebung des Ausgangsdokuments immun, da sie eine komplette WWW-Adresse umfasst und sozusagen »unten« ansetzt. Ein Verschieben des Quelldokuments wirkt sich dementsprechend nicht aus.

  Beispiel für eine absolute Pfadangabe:

  ```
  <a href="http://www.beispiel.de/presse.html">Presseseite</a>
  ```

  Eine absolute Pfadangabe umfasst das Übertragungsprotokoll, den Zielserver und das Zielverzeichnis sowie das eigentliche Zieldokument.

- Eine **wurzelbezogene Pfadangabe**
  bezieht sich auf die unterste Ebene der Webpräsenz als Ausgangspunkt.

  Die wurzelbezogene Pfadangabe ist vom Prinzip her vergleichbar mit einer absoluten Adressierung, bei der man die Domainadresse weglässt.

  Beispiel für eine wurzelbezogene Pfadangabe:

  ```
  <a href="/presse/index.html">Zur Presseseite</a>
  ```

  Eine wurzelbezogene Pfadangabe wird stets mit dem Slash eingeleitet. Vorteil gegenüber einem relativen Link ist, dass die Position des Ausgangsdokuments beliebig ist (sofern es innerhalb der gleichen Site bleibt), da der Weg immer »von der Wurzel aufwärts« zum Zieldokument führt. Das Ausgangsdokument kann sich hierarchisch über oder unter dem Zieldokument befinden oder verschoben werden, ohne dass der Link angepasst zu werden braucht.

Verbotene Zeichen in URL-Strings

Innerhalb von URL-Strings dürfen nur ASCII-Zeichen eingesetzt werden; verbotene Zeichen (beispielsweise das Leerzeichen oder Umlaute) müssen URL-encoded werden. Auch ist Vorsicht mit dem Zeichen »&« geboten, das innerhalb von Querystrings benötigt wird – hier empfiehlt sich wegen dessen Steuerzeichenfunktion in HTML seine Ersetzung durch ein Entity `&`. (Dies ist nicht nötig, wenn ein URL direkt in das URL-Feld des Browsers eingegeben wird, sondern nur da, wo er wie hier im Rahmen des Attributwertes im Quelltext erscheint).

```
<!-- Dies gibt ein Problem: -->
<a href="beispiel.cgi?kapitel=1&abschnitt=2">...</a>
```

ist daher besser zu schreiben als

```
<!-- Dies ist die Lösung: -->
<a href="beispiel.cgi?kapitel=1&amp;abschnitt=2">...</a>
```

Barrierefreiheit

Hyperlinks stellen prinzipiell keine Barriere dar, solange sie über die Tastatur bedienbar sind. Dies ist grundsätzlich der Fall, da Hyperlinks stets Teil der Tabindex-Reihe sind (lediglich der Internet Explorer bietet die Möglichkeit, Links aus dem Tabindex zu entfernen – nutzen Sie dies nicht!).

Die verlinkte Ressource sollte stets über das `href`-Attribut bezeichnet und nicht (bzw. nicht allein) mittels JavaScript-Eventhandler angesteuert werden (beispielsweise durch das Öffnen von Popup-Fenstern). Textbasierte Browser versagen an dieser Stelle. Eine Möglichkeit, das Öffnen von Popups (bzw. neuen Fenstern) barrierefreier zu gestalten wäre etwa folgende:

```
<a href="beispiel.html"
 onclick="window.open(this.href); return false;"
 title="Link öffnet sich in neuem Fenster">
 Information in neuem Fenster</a>
```

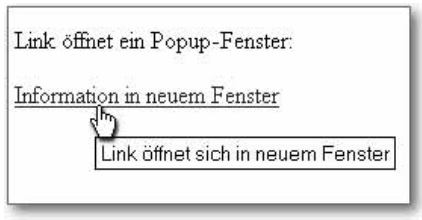

Abbildung 14.2 Link mit title-Attribut-Tooltip

Beachten Sie das Statement `return false` im Eventhandler, das die Weitergabe des Klick-Events an das `href`-Attribut unterbindet. Solange JavaScript aktiviert (bzw. verfügbar) ist, öffnet der Link sich über `window.open(this.href)` in einem neuen Fenster, wobei `this.href` den Wert des `href`-Attributs übergibt. Sie können alternativ auch eine Funktion aufrufen, die bestimmte Fenstereigenschaften (Größe, Position) erzwingt. Ist JavaScript nicht verfügbar, tritt das `href`-Attribut in Kraft und öffnet die Ressource im aktuellen Fenster. Das `title`-Attribut gibt im Vorfeld Auskunft über das Öffnen eines Fensters. Beachten Sie, dass das `target`-Attribut für *strict*-DTDs nicht zur Verfügung steht.

Eventhandler, die zur Steuerung von Scripten eingesetzt werden, sollten neutral sein wie `onclick` oder `onfocus` und nicht gerätebezogen wie `onkeydown` oder `onmouseover`. Einige Screenreader besitzen durchaus Möglichkeiten, Mausevents über die Tastatur auszulösen, und können mit Popup-Fenstern umgehen, jedoch ist dies nicht die Regel.

Problematisch sind auch **Links ohne Textinhalt**, beispielweise solche, die nur eine Grafik ohne `alt`-Attribut enthalten (Letzteres ist ohnehin bereits ein Fehler). Sorgen Sie stets dafür, dass für Textbrowser oder Screenreader auswertbare Inhalte im `<a>`-Container enthalten sind.

Verwenden Sie aussagekräftige Linktexte, die das Ziel beschreiben, und vermeiden Sie Stereotypien wie »Hier klicken«, »Weiter« oder »Mehr«. Vermeiden Sie auch das Wort »Link« am Anfang des Linktextes, da **Screenreader** in der Regel vor dem Linktext das Wort »LINK« als Signal einfügen. Ein weiterer Grund für aussagekräftige Linktexte besteht darin, dass Screenreader ihren Nutzern eine Liste aller im Text vorkommenden Verweise außerhalb ihres Kontexts zur Verfügung stellen können, um ihnen eine schnellere Navigation zu ermöglichen.

Wenn Sie Links für ein horizontales Menü nebeneinander platzieren, achten Sie darauf, dass Sie die einzelnen Links durch **Trennzeichen** (»delimiter«) wie »|« voneinander abheben, da Screenreader sonst die Einzellinks nicht adäquat vermitteln können. Für **Breadcrumb-Navigationen** können Sie in dieser Funktion auch »/« oder »>« einsetzen (Abbildung 14.3).

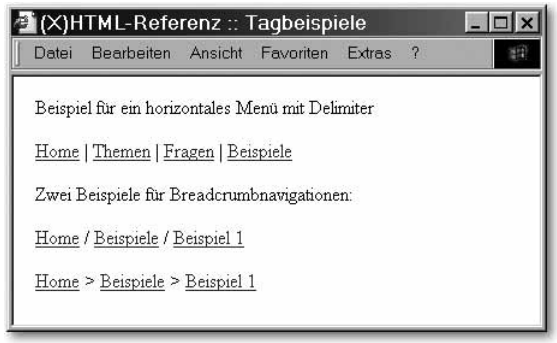

Abbildung 14.3 Menüs mit Delimiterzeichen

14.2 abbr

Logische Auszeichnung: Der `<abbr>`-Container signalisiert, dass der in ihn eingeschlossene Text als Abkürzung (*abbreviation*) eines Begriffs zu verstehen ist.

14.2.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Das Element darf überall in Block- und Inlineelementen auftreten.
Erlaubte Inhalte	Text (`#PCDATA`) und Inlineelemente
Offizielle W3C Spezifikationen	HTML 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile iMode XHTML 1.0, XHTML mobile profile **abgeschafft** in iMode XHTML 1.1

Syntaxbeispiel:

```
<abbr> ... </abbr>
```

Defaultrendering:
Das Element ist ein unsichtbarer Marker.

CSS-Umsetzung des HTML-Defaults:

```
abbr            { display:inline }
```

mit title-Attribut:

```
abbr[title]     { display:inline;
                  border-bottom:dotted 1px inherit; }
```

14.2.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	– (Element wird im IE derzeit nicht unterstützt)

Elementspezifische Attribute

W3C	keine

14.2.3 Erläuterungen zum Element

Das Element `<abbr>` dient der logischen Auszeichnung seines Inhalts als Abkürzung eines Begriffes. Der Container besitzt ohne Verwendung von Attributen (es stehen die Standardattribute zur Verfügung) keine optische Präsentation.

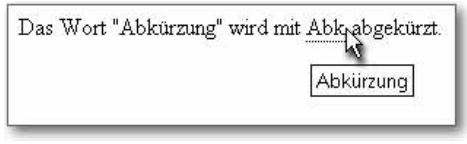

Abbildung 14.4 Das Element <abbr> in Mozilla (hier mit title-Attribut)

Sinnvoll ist die Verwendung des `title`-Attributs, um den abgekürzten Begriff bei Mouseover als Tooltip anzuzeigen (s. Abb.):

```
<p>Das Wort "Abkürzung" wird mit
<abbr title="Abkürzung">Abk.</abbr> abgekürzt.</p>
```

Das Element <abbr> und Barrierefreiheit

Ein Screenreader ist zudem in der Lage (mit der Funktion »Abkürzungen und Akronyme erweitern«), anstelle des Taginhalts den Titeltext selbst auszugeben.[6] Dies ist besonders interessant unter dem Aspekt, dass diejenigen Abkürzungen, die zwar problemlos gelesen, nicht aber ausgesprochen werden können, durch verständliche, den Textfluss nicht behindernde Ausgaben ersetzt werden können.

```
<p>Dies wird <abbr title="zum Beispiel">z.B.</abbr> hier
deutlich.</p>
```

Eine **Betitelung** von Abkürzungen ist daher im Sinne der Barrierefreiheit grundsätzlich wünschenswert. Je nach Einstellung des Screenreaders (**Expand abbreviations**) wird die Abkürzung oder der durch sie ersetzte Text verlesen:

»Dies wird *zum Beispiel* hier deutlich« **[PAUSE]**[7]

Bei Anwesenheit des `title`-Attributs hat sich, sofern das Element vom Browser unterstützt wird, eine Präsentation mit Unterpunktierung in Schriftfarbe als gängige Praxis etabliert. Ohne `title`-Attribut ist das Element in der Seite nicht sichtbar – ein `title`-Text könnte aber nachträglich per Script hinzugefügt werden (aus einer Datenbank o.ä.). Abkürzungen auch ohne `title` auszuzeichnen, kann daher dennoch sinnvoll sein.

Bei zu häufigem Auftreten ein und derselben Abkürzung oder allgemein bekannter Abkürzungen (z. B. der Titel »Dr.« in einem Adressverzeichnis) mag die Unterpunktierung optisch lästig wirken. In diesem Falle sollte man jedoch im Hinblick auf Screenreadernutzer nicht auf das `title`-Attribut verzichten, sondern dessen Präsentation mittels CSS deaktivieren:

```
abbr {border: 0 }
```

6 Abhängig vom verwendeten Screenreader – die beschriebene Funktion steht beispielsweise bei IBMs »Home Page Reader« zur Verfügung (siehe: www-3.ibm.com/able/solution_offerings/hpr.html) oder bei Dolphins HAL (siehe: www.yourdolphin.com).

7 Ein Punkt am Satzende wird normalerweise nicht vorgelesen, sondern eine Pause gelassen.

Bei gebräuchlichen Abkürzungen kann man davon ausgehen, dass ein Screenreader über eine entsprechende Bibliothek verfügt und auch ohne verwendetes <abbr>-Element den nicht abgekürzten Begriff einsetzt.

Hinweis: Der Tag <abbr> wird oft fälschlich als inhaltlich gleichbedeutend mit <acronym> (s.d.) betrachtet. Bei einem **Akronym** handelt es sich jedoch nicht um eine Abkürzung, sondern um ein Kunstwort, das aus den Anfangsbuchstaben einer mehrwortigen Bezeichnung gebildet wird – so stellen beispielsweise »NASA«, »WWW« oder »HTML« keine Abkürzungen dar, sondern sind Akronyme. **[«]**

Der Internet Explorer unterstützt das Element <abbr> erst ab dessen Version 7.0. Im Internet Explorer bis 6.0 wird das Element vollständig ignoriert, es wird also weder ein title-Tooltip gezeigt noch an <abbr> gebundene Styleanweisungen oder Events ausgeführt (es lässt sich daher in älteren IE nicht einfach eine optische Konsistenz zu anderen Browsern erreichen, es sei denn, man deaktiviert die Unterpunktierung generell). Mit id-Attribut versehene <abbr>-Elemente können jedoch auch im IE vor 7.0 angesprochen werden (mit getElementById()), sofern dies erforderlich ist. Ebenso funktioniert die Ansprache der <abbr>-Collection über getElementsByTagName().

14.3 acronym

Logische Auszeichnung: Das Element <acronym> dient zur Auszeichnung von Akronymen[8] innerhalb eines Dokuments.

14.3.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text (#PCDATA)

8 Zur Begriffsverwirrung zwischen Akronym und Abkürzung siehe die Ausführungen beim Element <abbr>.

Offizielle W3C Spezifikationen	HTML 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile iMode XHTML 1.0, XHTML mobile profile **abgeschafft** in iMode XHTML 1.1

Syntaxbeispiel:

```
<acronym> ... </acronym>
```

Defaultrendering:
Das Element ist ein unsichtbarer Marker.

CSS-Umsetzung des HTML-Defaults:

```
acronym         { display:inline }
```

mit title-Attribut:

```
acronym [title] { display:inline;
                  border-bottom:dotted 1px inherit; }
```

14.3.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `onfocus`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	keine
Internet Explorer	*Access-, Edit-Attribute*

14.3.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character`

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number`

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"` (IE 5)

`unselectable = "on" | "off"` (IE 5)

14.3.4 Erläuterungen zum Element

Das Element `<acronym>` dient der logischen Auszeichnung seines Inhalts als Akronym. Der Container besitzt ohne Verwendung von Attributen (es stehen die Standardattribute zur Verfügung) keine optische Präsentation.

Abbildung 14.5 Das Element <acronym> in Mozilla (mit title-Attribut)

Sinnvoll ist die Verwendung des `title`-Attributs, um den abkürzten Begriff bei Mouseover als Tooltip anzuzeigen (siehe Abbildung 14.5):

```
<p>Der Begriff
<acronym title="Hypertext Markup Language">HTML</acronym>
ist ein Akronym für "Hypertext Markup Language".</p>
```

Ein Screenreader ist (vergleichbar mit der Funktionalität für den `<abbr>`-Tag) ebenfalls dazu in der Lage, für Akronyme anstelle des Taginhalts den Titeltext auszugeben. Dies ist nicht generell sinnvoll; unterschieden werden sollte zwischen allgemein bekannten Akronymen, die im täglichen Sprachgebrauch wie

ein feststehender Begriff verwendet werden, und solchen, die mehrdeutig[9] oder weniger verbreitet sind und deren Bedeutung sich nicht aus dem Kontext erschließen.

Bei Einsatz eines Screenreaders könnte sich je nach dessen Einstellung im obigen obendrein eine unerwünschte Redundanz ergeben. Ein Screenreader könnte jedoch das Akronym (und anschließend den weiterlaufenden normalen Text) auch einfach ansagen:

»Der Begriff **AKRONYM** HTML **NORMALE BEDEUTUNG** ist ein Akronym für Hypertext Markup Lanuage« **[Pause]**

Eine Alternative zur Verwendung des `title`-Attributs bestünde darin, die Erläuterung eines Akronyms bei seinem ersten Auftreten im Dokument ausgeschrieben beizufügen, also beispielsweise »HTML (Hypertext Markup Language)« oder auch umgekehrt »Hypertext Markup Language (HTML)«. Hiermit wären die Bedingungen der Barrierefreiheit gemäß BITV bereits erfüllt.

Bei Anwesenheit des `title`-Attributs hat sich in einigen Browsern, vergleichbar mit der Praxis bei `<abbr>`, die Präsentation mit Unterpunktierung in Schriftfarbe etabliert. Der Internet Explorer folgt dem nicht, was jedoch durch eine entsprechende explizite Styleanweisung (halbwegs) korrigiert werden kann:

```
<p>Der Begriff <acronym title="Hypertext Markup Language"
style="border-bottom:dotted 1px inherit">HTML</acronym> ist ein
Akronym für "Hypertext Markup Language".</p>
```

Die generelle Betitelung aller in einer Seite vorkommenden Akronyme erscheint nicht sinnvoll, wenn sich ein Begriff beispielsweise häufig (z. B. mehrfach in aufeinanderfolgenden Zeilen) wiederholt; auch hier könnte ein Script hilfreich sein, das nur – wenn überhaupt – das erste Auftauchen mit einem `title`-Attribut versieht.

Auch die anvisierte Zielgruppe sollte in Bezug auf die Frage »Betitelung oder nicht« in Betracht gezogen werden. Ob beispielsweise in einem Text, der sich an Webdesigner richtet, das Akronym »HTML« unbedingt erläutert werden muss, braucht sicher kaum diskutiert zu werden …

9 So hat »CSS« weltweit rund 146 verschiedene Bedeutungen. Im Allgemeinen wird aus dem Kontext hervorgehen, welche davon gemeint ist – andernfalls ist eine Erläuterung sinnvoll. Die Website http://www.acronymfinder.com/ gibt hier gute Auskunft.

14.4 address

Logische Auszeichnung: Das Blockelement `<address>` kennzeichnet eine Adresse oder Kontaktmöglichkeit (ursprünglich in Zusammenhang mit dem jeweiligen Dokument).

14.4.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	Text
Kontext	Das Element darf unmittelbar innerhalb des `<body>`-Containers und innerhalb folgender Elemente auftreten: `<blockquote>`, `<button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<noscript>`, `<object>`, `<td>`, `<th>`.
Erlaubte Inhalte	Text (`#PCDATA`), Inlineelemente und bestimmte Blockelemente (`<p>`, ` `)
Offizielle W3C Spezifikationen	HTML Level 0; HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile iMode XHTML 1.0, XHTML mobile profile **abgeschafft** in iMode XHTML 1.1

Syntaxbeispiel:

```
<address> ... </address>
```

Defaultrendering:
Das Element wird als Blockelement und sein Inhalt kursiv dargestellt.

CSS-Umsetzung des HTML-Defaults:

```
address    { display:block;
             font-style: italic;
             }
```

14.4.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `onfocus`, `onbeforecopy`, `onbeforecut`, `onbeforepaste`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	keine
Internet Explorer	*Access-, Edit-Attribute*

14.4.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character`

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number`

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"` (IE 5)

`unselectable = "on" | "off"` (IE 5)

14.4.4 Erläuterung zum Element

Das Element `<address>` dient der logischen Auszeichnung seines Inhalts als Adresse. Der Inhalt des Blockelements wird üblicherweise kursiv dargestellt.

Der Wert des `title`-Attributs wird in allen aktuellen Browsern als **Tooltip** angezeigt (s. Abb.), seine An- oder Abwesenheit hat ansonsten auf die Präsentation des Elements keinen weiteren Einfluss.

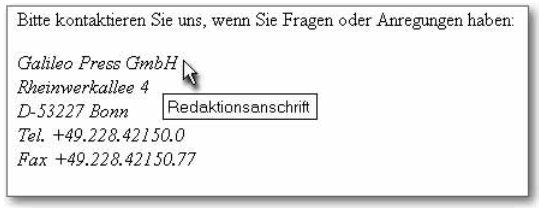

Abbildung 14.6 Das Element <address> in Mozilla (mit title-Attribut)

Beispiel: **[zB]**

```
<address title="Redaktionsanschrift">
Galileo Press GmbH<br />
Rheinwerkallee 4<br />
D-53227 Bonn<br />
Tel. +49.228.42150.0<br />
Fax +49.228.42150.77<br />
</address>
```

14.5 applet

Das Element `<applet>` dient dazu, ein Java-Applet in eine HTML-Seite einzubetten. Von der Verwendung wird durch das W3C zugunsten des vielseitigeren Elements `<object>` abgeraten.

14.5.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Applet (Deprecated)
Kontext	Das Element darf überall in Block- und Inlineelementen auftreten.
Erlaubte Inhalte	Elemente `<param>` zur Übergabe von Werten an das Applet, Block- und Inlineelemente sowie Text (`#PCDATA`) als alternativ darzustellender Inhalt.
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0 tr./fs, 4.01 tr./fs, XHTML 1.0 tr./fs **abgeschafft** in HTML 4.01 strict, XHTML 1.0 strict, XHTML 1.1, 2.0
Mobil-Spezifikationen	iMode XHTML 1.0 **nicht enthalten** in HTML 4.0 mobile, iMode XHTML 1.1, XHTML mobile profile

Syntaxbeispiel:

```
<applet [Attribute]> ... </applet>
```

14.5.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncellchange`, `oncontextmenu`, `oncut`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onfocus`, `onhelp`, `onload`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onrowsdelete`, `onrowsinserted`, `onscroll`

Elementspezifische Attribute

W3C	`align`, `alt`, `archive`, `code`, `codebase`, **`height`**, `hspace`, `name`, `object`, `vspace`, **`width`**
Internet Explorer	`src` *Access-, Edit-, Databinding-Attribute*
Netscape	`mayscript` *(NN4)*

14.5.3 Offizielle Attribute – W3C

align

Wert	*"top"* | *"middle"* | *"bottom"* | *"left"* (float) | *"right"* (float)
Typ	festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut erlaubt es, ein eingebettetes Applet gleich einem Image im Zeilenkontext oder mittels der Float-Attributwerte `"left"` und `"right"` am entsprechenden Rand des übergeordneten Contai-

ners zu positionieren. Zusätzlich zu den Werten der Spezifikation unterstützen einige Browser auch die Werte `"center"`, `"baseline"`, `"absbottom"`, `"absmiddle"` und `"texttop"`, von deren Einsatz jedoch abzuraten ist.

alt

Wert	*cdata*
Typ	alternative Textbeschreibung
Verwendung	Optional

Barrierefreiheit: Das `alt`-Attribut dient der Bindung eines Alternativtextes an das Element, der dargestellt werden kann, falls Applets nicht unterstützt werden oder das Applet nicht geladen wird.

archive

Wert	*URL [, URL2, URL3, ... URLn]*
Typ	ein oder mehrere URI-Strings
Verwendung	Optional

Das `archive`-Attribut kann den absoluten oder relativen URL einer alternativen, gepackten Version des Applets (GNU-Zip-File, unkomprimiert) nennen, um den Ladevogang zu beschleunigen. Zusätzlich können die URLs weiterer, zu ladender Klassen angegeben werden.

code

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional

Das Attribut `code` nennt den absoluten oder relativen URL der als Applet zu ladenden Javaklasse mit der Dateiendung `.class`. Obwohl das `code`-Attribut die gängigste Methode ist, das Applet einzubinden, kann dies alternativ auch über das `archive`- oder das `object`-Attribut erfolgen. Das Attribut ist *case-sensitive*.

codebase

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional

Das `codebase`-Attribut gibt einen absoluten oder relativen URL an, der als Basis-URL für den URL des `code`-Attributs dient. Ein relativer URL wird als bezogen auf den URL des Dokuments oder, falls ein solcher vorhanden ist, auf den `href`-Wert des `<base>`-Tags ausgewertet.

height

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Obligatorisch

Das `height`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für die Höhe des für das Applet freizuhaltenden Raums.

hspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut). Das `hspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch links und rechts vom Applet freizuhaltenden Raum.

name

Wert	*nmtoken*
Typ	Zeichenkette in Form eines Nametokens
Verwendung	Optional

Das `name`-Attribut nennt eine Zeichenkette, die von Scripten als Bezeichner für das Applet-Objekt verwendet werden kann. Alternativ kann das `id`-Attribut eingesetzt werden. Das Attribut ist *case-sensitive*.

object

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional

Das `object`-Attribut nennt einen URL, unter dem das Applet in serialisierter Form (Klasseninformation und gespeicherter Zustand des Applets als Bytefolge) zu finden ist. Das Attribut ist *case-sensitive*.

vspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `vspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch über und unter dem Applet freizuhaltenden Raum.

width

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Obligatorisch

Das `width`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe; die Breite des für das Applet zu reservierenden Raums.

14.5.4 Proprietäre Attribute – Internet Explorer

src

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional
Einführung	IE 4.0

Das `src`-Attribut legt einen URL für den Download des Applet-Files fest.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character`

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number`

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false" | "true"` (IE 5)

`unselectable = "on" | "off"` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

14.5.5 Proprietäre Attribute – Netscape

mayscript

Wert	*"mayscript"*
Typ	festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

(Nur Netscape 4.x) Durch Setzen des Attributs `mayscript` gestattet man Netscape-Browsern, das eingebettete Applet per Script zu steuern. Von anderen Browsern wird diese Funktionalität nicht unterstützt.

14.5.6 Erläuterung zum Element

Das Element `<applet>` wurde von Netscape Ende 1995 eingeführt, um die Einbettung von Java-Applets in HTML-Dokumente vergleichbar mit der Einbettung von Bildern durch den `<img/>`-Tag zu ermöglichen. In HTML 3.2 erstmals spezifiziert, wird der Tag seitdem durch alle verbreiteten Browser unterstützt, eine entsprechende Java-Laufzeitumgebung auf dem Rechner vorausgesetzt.

Applets werden eingeschlossen von den Tags `<applet> ...</applet>`. Drei Angaben im `<applet>`-Tag sind obligatorisch: `code`, `width`, `height`. Das Attribut `code="appletname"` gibt den Namen des Applets an. Die Attribute `width` und `height` beziehen sich (genau wie bei einem Bild) auf Breite und Höhe der Darstellung in Pixeln.

Der einfachste Aufruf im Code sieht folgendermaßen aus:

```
<applet code="Test.class" width="100" height="120"></applet>
```

Weitere Angaben sind möglich, zu denen `codebase`, `alt`, `align`, `vspace` und `hspace` zählen. Die drei letzten Argumente haben hier die gleiche Funktion wie etwa im `<img/>`-Tag. Das Attribut `alt` enthält Text, der gezeigt wird, wenn der Browser zwar den Applet-Tag versteht, aber Java nicht ausführen kann.

Das Argument `codebase` gibt einen URL an, unter dem das Applet zu finden ist. Normalerweise ist dies einfach eine relative Verzeichnisangabe, es kann sich aber auch um eine absolute Web-Adresse handeln. Fehlt dieses Argument, dann wird einfach das derzeit aktive Verzeichnis als Quelle angenommen. Interessant ist `codebase` vor allem deswegen, weil damit auf beliebige Websites zugegriffen werden kann, die Applets enthalten.

Um einem eingebetteten Applet Informationen übergeben zu können, darf der `<applet>`-Container einen oder mehrere `<param>`-Tags enthalten, deren Attribute Namen und Wert des Parameters nennen. Diese Elemente werden ignoriert, falls das Applet nicht ausgeführt werden kann oder der Browser den `<applet>`-Tag nicht unterstützt:

```
<applet code="minesweeper.class" archive="minesweeper.zip"
 height="200" width="200" align="left">
   <param name="modus" value="einsteiger" />
   <param name="spielspeicher" value="false" />
</applet>
```

Falls das über das `code`-Attribut bezeichnete Applet nicht geladen oder ausgeführt werden kann, soll (zumindest in der Theorie) der alternative Elementinhalt dargestellt werden, den der `<applet>`-Container neben den `<param>`-Elementen besitzen darf (das Seitenlayout dürfte hiervon nicht unbeeinträchtigt sein, da in diesem Falle die Abmessungen des Bereiches nicht definiert sind). Sofern das Applet ausgeführt wird, ist die Darstellung dieser Inhalte unterdrückt:

```
<applet code="minesweeper.class" archive="minesweeper.zip"
 height="200" width="200" align="left">
   <param name="modus" value="einsteiger">
   <b>Schade, Sie haben kein Java installiert.</b>
</applet>
```

Seit HTML 4.0ff gilt das Element als *deprecated* und ist nur noch in *transitional*- und *frameset*-Varianten von HTML 4.01 und XHTML 1.0 gestattet – an seiner Stelle wird die Verwendung von `<object>` empfohlen. In XHTML 1.1 ist `<applet>` abgeschafft.

Barrierefreiheit

Da alternative Endgeräte Applets nicht in jedem Fall darstellen können, sollte zumindest ein beschreibender Text im `alt`-Attribut zur Verfügung gestellt werden. Auch bei der Erstellung des Alternativinhalts des Elements (siehe oben) können Aspekte der Barrierefreiheit berücksichtigt werden.

Es muss darauf geachtet werden, dass die Seite ihren vollen Informationswert behält, wenn Applets nicht unterstützt oder absichtlich abgeschaltet werden (WAI 6.3, BITV 7.3).

Beim Einsatz von Applets muss sichergestellt sein, dass unabhängig vom Eingabegerät navigiert werden kann. So dürfen keine gerätespezifischen Event-Handler eingesetzt werden, sondern ausschließlich logische, geräteunabhängige wie `onfocus`, `onblur` (WAI 9.2 und 9.3, BITV 6.4, 9.2, 9.3).

14.6 area

Das `<area>`-Element dient als Unterelement des `<map>`-Containers dazu, die anklickbaren Bereiche einer clientseitigen Imagemap zu definieren.

14.6.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Leeres Element
XHTML-Modul	Client-side Image Map
Kontext	Das `<area>`-Element tritt als Kindelement des `<map>`-Containers auf.
Erlaubte Inhalte	Keine (leeres Element).
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01, XHTML 1.0, 1.1, XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile, iMode XHTML 1.0 **abgeschafft** in iMode XHTML 1.1, XHTML mobile profile

Syntaxbeispiel:

`<area [Attribute]>` (für valides HTML: kein Endtag gestattet)

`<area [Attribute]/>` (in XHTML)

Defaultrendering:

Das Element wird nicht dargestellt, sondern dient als Teil einer Imagemap-Definition.

CSS-Umsetzung des HTML-Defaults:

```
area   { display:none }
```

14.6.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	`onblur`, `onfocus`
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`accesskey`, **`alt`**, `coords`, **`href`**, `nohref`, `shape`, `tabindex`, `target` (XHTML Mod: Target),
Internet Explorer	`notab`[10] *Access-, Edit-Attribute*

14.6.3 Offizielle Attribute – W3C

accesskey

Wert	*character*
Typ	Alphanumerisches Zeichen, nicht *case-sensitive*.
Verwendung	Optional

Barrierefreiheit: Das Attribut `accesskey` legt im Sinne der Accessibility ein (einzelnes) Tastaturzeichen fest, das als Keyboard-Shortcut bei gleichzeitig gedrückter Alt-Taste den Link ausführt.

10 Abgeschafft.

Vorsicht bei Accesskeyduplikaten

Besitzen mehrere Links den gleichen Accesskey, so wird derjenige ausgeführt, der im Quelltext als erster erscheint (alle anderen können nicht über den Accesskey angesprochen werden). Bei der Konkurrenz zwischen Imagemaplinks und anderen Hyperlinks ist dabei die Position der Imagemap-Grafik im Dokument ausschlaggebend, nicht die der Map-Definition (also des `<area>`-Elements).

alt

Wert	*cdata*
Typ	Eine beliebige CDATA-Zeichenkette
Verwendung	Obligatorisch

Barrierefreiheit: Das `alt`-Attribut stellt für den Fall, dass keine Bilder dargestellt werden[11], einen alternativen, das Linkziel beschreibenden Text zur Verfügung.

[»] *Hinweis:* Netscape 4.x stellt den Wert des Attributs als Tooltip dar – dies wird von aktuellen Browsern nicht (mehr) so gehandhabt. Stattdessen kann das `title`-Attribut für diesen Zweck eingesetzt werden.

coords

Wert	*x_koordinate1, y_koordinate1 [, x_koordinate2, y_koordinate2 ...]*
Typ	Eine Folge (meist Pärchen) kommaseparierter Ganzzahlenwerte
Verwendung	Optional

Die in `coords` übergebenen Werte werden als Koordinatenwerte (x, y) der Begrenzungen eines anklickbaren Bereichs einer Imagemap interpretiert. Das Attribut wird gewöhnlich gemeinsam mit dem `shape`-Attribut verwendet – ist das `shape`-Attribut nicht gesetzt, wird ein rechteckiger Bereich angenommen.

Die **Interpretation** der Werte hängen vom `shape`-Attribut wie folgt ab:

- **Rechteckform (als vier Koordinatenwerte)**
  `shape="rect"` bzw. kein `shape`-Attribut (gel. auch: `"rectangle"`,)
  `coords="`x_linksoben, y_linksoben, x_rechtsunten, y_rechtsunten`"`
- **Geschlossenes Polygon (für N-Eckpunkte: als N x 2 Koordinatenwerte)**
  `shape="poly"` (gelegentlich auch: `shape="polygon"`)
  `coords="`x_p1, y_p1, x_p2, y_p2, x_p3, y_p3, ... x_pN, y_pN`"`

11 Dies gilt leider nicht für den Fall eines »broken image«.

- **Kreisform (als 2 Koordinatenwerte + Radiuswert)**
  `shape="circle"` (gelegentlich auch: `shape="circ"`)
  `coords="`x_mittelpunkt, y_mittelpunkt, pixel_radiuswert`"`
- **Defaultform (keine Koordinateninterpretation)**
  `shape="default"` (entspricht nicht weggelassenem `shape`-Attribut!)
  `coords` kann entfallen bzw. wird ignoriert

Das folgende Beispiel definiert einen kreisförmigen, anklickbaren Bereich von 40 Pixel Durchmesser, dessen Mittelpunkt je 25 Pixel rechts und unterhalb des rechten oberen Ecks des Bezugsobjekts liegt.

```
<map name="kreis">
   <area href="beispiel.html"
    shape="circle" coords="25, 25, 20" />
</map>
```

href

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional, bei Abwesenheit von `nohref` obligatorisch

Enthält den absoluten oder relativen URL der verlinkten Ressource.

Hinweis: Wenn kein `nohref`-Attribut vorhanden ist, ist das `href`-Attribut **obligatorisch**. **[«]**

nohref

Wert	*"nohref"*
Typ	festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `nohref`-Attribut wird anstelle des `href`-Attributs eingesetzt, wenn ein aktiver Bereich zwar definiert, diesem aber keine zu verlinkende Ressource zugeordnet werden soll.

Hinweis: Der Bereich spielt insofern eine Rolle, als dass das `title`-Attribut bei Mouseover als Tooltip dargestellt wird. Der Mauszeiger bleibt dabei (korrekterweise) in seiner Pfeilform. **[«]**

Besonderheit bei Mozilla/Firefox

In Mozilla/Firefox ist der inaktive Bereich stets dennoch in der normalen Tab-Reihenfolge über Tabulator ansprechbar, in Opera und Internet Explorer nur dann, wenn für ihn gleichzeitig ein `tabindex`-Wert vergeben ist (sonst wird der Bereich beim Tab-Select übersprungen).

shape

Wert	*"rect"* | *"circle"* | *"poly"* | *"default"* (| *"rectangle"* | *"circ"* | *"polygon")*
Typ	festgelegte Bezeichner
Verwendung	Optional

Das `shape`-Attribut nennt die Form des darzustellenden, anklickbaren Bereichs und beeinflusst so die Interpretation der mit dem `coords`-Attribut übergebenen Koordinatenwerte.

Ist kein `shape`-Attribut gesetzt, so tritt dessen **Defaultwert** `"rect"` in Kraft.

Ist ein Bereich auf Formfaktor `"default"` gesetzt (in diesem Fall ist kein `coords`-Attribut erforderlich), so gilt dieser für die gesamte Fläche der Imagemap-Grafik, sofern nicht für Teilflächen andere Bereiche Vorrang haben (mit »Defaultform« ist also nicht der »Defaultwert« gemeint!).

Definitionsreihenfolge relevant bei überlappenden Bereichen

Die Reihenfolge der Area-Definitionen ist *relevant*, sofern sich die Bereiche überlappen. Für die Überschneidungsfläche zweier in diesem Sinne konkurrierender Bereiche gilt die zuerst erfolgte Definition als maßgeblich. Will man `shape="default"` in einer Imagemap einsetzen, ist es daher sinnvoll, diese Definition als letzte zu vereinbaren, da sonst danach vereinbarte, andere Bereiche »verdeckt« werden (über Tab-Select bleiben sie zugänglich). Andererseits addieren sich mehrere Bereiche gleichen Linkziels, so dass sich auch ohne die Verwendung von `"poly"` komplexe Umrisse definieren lassen.

tabindex

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Optional

Barrierefreiheit: Das Attribut `tabindex` bestimmt die Reihenfolge, in der Hyperlinks nach dem Laden der Seite über die Tabulatortaste (⇆-Taste) angesprochen (in Fokus gebracht) werden können. Die anklickbaren Bereiche, die zu den mit

Tabindex versehenen <area>-Elementen korrespondieren, werden in aufsteigender Reihenfolge entsprechend der vergebenen Werte aktiv (d.h. tabindex="5" nach tabindex="3", auch wenn kein dazwischenliegender Wert vergeben ist).

Die Tab-Reihenfolge gestaltet sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes tabindex-Attribut in Quelltextreihenfolge. (Zu den aktivierbaren Elementen gehören allgemein neben <area> auch <a>, <button>, <input>, <object>, <select> und <textarea>.)

Für den Fall, dass zwei <area>-Elemente den gleichen Index erhalten, wird der Bereich desjenigen von beiden zuerst in Fokus genommen, das im Quelltext als erstes steht, anschließend der des anderen (der zweite Bereich wird nicht übersprungen).

Hinweis: Im Falle der Konkurrenz gleicher Tabindex-Werte zwischen Bereichen einer Imagemap und anderen Elementen (Hyperlinks oder Formularfeldern) ist nicht die Quelltextposition des <area>-Elements, sondern die des als Imagemap dienenden Bildes ausschlaggebend. **[«]**

target

Wert	*nmtoken*
Typ	festgelegte und frei wählbare Bezeichner
Verwendung	Optional

Das target-Attribut verwendet einen freigewählten Bezeichner eines (existierenden) Browserfensterobjekts oder – im Framesetkontext – einen Bezeichner eines benannten Frameobjekts bzw. einen der vordefinierten Bezeichner "_self", "_parent", "_top" oder "_blank".

Anmerkung: In XHTML ist das target-Attribut Teil des Moduls "target".

Der Hyperlinkbefehl wird in dem über das target-Attribut bezeichneten Fenster oder Frame ausgeführt:

- _self
  Der Link wird im gleichen Fensterobjekt des aktuellen Framekontexts ausgeführt.

- `_parent`
  Der Link wird im unmittelbar dem aktuellen Frame übergeordneten Frameobjekt des aktuellen Framekontexts ausgeführt.
- `_top`
  Der Link wird im hierarchisch obersten Frame des aktuellen Framekontexts ausgeführt (auch als »Hauptfenster« bezeichnet; der Frameset wird entladen).
- `_blank`
  Der Link wird in einem sich neu außerhalb des aktuellen Framekontexts öffnenden Fenster ausgeführt.
- Ein frei gewählter Bezeichner
  führt den Link im gleichnamigen Fensterobjekt aus (falls vorhanden – dies kann ein benanntes Frameobjekt oder ein frei schwebendes Fenster sein) oder öffnet ein Browserfenster, das anschließend unter diesem Bezeichner ansprechbar ist.

[»] *Hinweis:* Der vordefinierte Bezeichner `_blank` öffnet ebenfalls ein neues Fenster, vergibt an dieses aber **keinen Bezeichner**, so dass dieses Fenster in Folge über Links nicht erneut ansprechbar ist.

Lokales Target hat Vorrang vor globalem Target

Ein explizit pro Link gesetztes Target hat Vorrang vor einem mit Hilfe des `<base>`-Elements und dessen `target`-Attribut gesetzten globalen Target.

14.6.4 Proprietäre Attribute – Internet Explorer

notab

Wert	–
Typ	leeres Marker-Attribut
Verwendung	Optional
Gültigkeit	IE 3; seit IE 4.0 **abgeschafft**

(Historisch) Das Setzen des `notab`-Attributs verhindert(e), dass der Bereich über die Tabulatortaste angesprochen werden kann. Es wird inzwischen auch vom Internet Explorer nicht mehr unterstützt.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (ab IE 5.5)

IE-Edit-Attribute
`unselectable = "on" | "off"` (IE 5)

14.6.5 Erläuterungen zum Element

Das `<area>`-Element dient als Unterelement des `<map>`-Containers dazu, die anklickbaren Bereiche einer clientseitigen **Imagemap** zu definieren.

Die Mapdefinition wird mit Hilfe des `name`-Attributs (in XHTML 1.1 des `id`-Attributs) des `<map>`-Containers in Form einer `ID-IDREF`-Verknüpfung an ein `usemap`-Attribut einer über den `<img>`-Tag eingebetteten **Grafik** oder eines über den `<object>`-Container eingebetteten beliebigen **Objekts** gebunden.

Abbildung 14.7 Kreisförmiger Bereich einer Imagemap, mit Tab-Select aktiviert

Ein `<map>`-Container kann eine beliebige Zahl von `<area>`-Definitionen enthalten, die gemeinsam die dem Zielobjekt zugeordnetete Imagemap erzeugen. Die Position der (nicht gerenderten und daher unsichtbaren) Imagemap-Definition im Quelltext ist dabei irrelevant (sie kann an beliebigen Stellen im Body innerhalb von Block- oder Inlineelementen platziert werden), die Reihenfolge der Area-Definitionen innerhalb der Map in bestimmten Fällen nicht.

Betroffen sind Überlappungen von zwei oder mehr Bereichen.

Überlappung von Bereichen gleichen Linkziels

Eine Überlappung ist ungefährlich, sofern die Bereiche dasselbe Linkziel haben; in diesem Fall *addieren* sich die Bereiche von ihrer Wirkung her, ungeachtet der Reihenfolge ihrer Definition.

Hierbei ist zu beachten, dass die Unterbereiche eines so zusammengesetzten Bereiches dennoch *einzeln* über Tab-Select angesprochen werden, sich in diesem Fall also *nicht* verhalten, wie ein einzelner (z. B. mit `shape="poly"` definierter) komplexer Bereich. Man kann sich behelfen, indem man allen `<area>`-Elementen, die Einzelbereiche erzeugen, ein `tabindex`-Attribut zuordnet, aber nur einem (beliebigen) `<area>`-Element eines zusammengesetzten Bereiches. Dies lässt die nicht indizierten Unterbereiche in die normale, erst anschließend berücksichtigte Tab-Reihenfolge zurückfallen.

Überlappung von Bereichen verschiedenen Linkziels

Im Überschneidungssegment zweier oder mehrerer aktiver Bereiche hat der in Quelltextreihenfolge **zuerst** vereinbarte Bereich Vorrang.

Überlappung aktiver und inaktiver Bereiche

Überschneiden sich ein aktiver Bereich und ein inaktiver `nohref`-Bereich, so hat ebenfalls der in Quelltextreihenfolge **zuerst** stehende Bereich Vorrang.

Es ist damit möglich, mittels eines `nohref`-Bereiches aus einem aktiven Bereich einen Teil »auszuschneiden« (eigentlich tritt eine Verdeckung auf). Durch Überlagerung eines aktiven und eines inaktiven Kreisbereichs kann so in der Summe ein ringförmiger aktiver Bereich erzeugt werden.

Bereich ganz oder teilweise außerhalb der Objektfläche

Es wird nur der Teil eines aktiven Bereichs berücksichtigt, der über der Objektfläche liegt, wobei hiermit dessen Inhaltsbereich (innerhalb der Border, zuzüglich eventuelles Padding) gemeint ist.

Der Borderbereich selbst und eventuelle Objektmargins (auch durch `vspace` oder `hspace` definierte) sind nicht aktivierbar; der Bereich wird dort abgeschnitten. Ein Bereich, der vollständig außerhalb der Objektfläche positioniert ist, wird nicht berücksichtigt.

Koordinatenursprung

Der Koordinatenursprung bezieht sich stets auf die linke obere Ecke des eigentlichen Objekts, auch wenn dessen aktivierbare Fläche durch Paddings vergrößert ist.

Anmerkung: In **XHTML 2.0** sind `<map>`- und `<area>`-Element *abgeschafft*; ihre Funktionalität wird vom dort neu hinzugekommenen `<nl>`-Container (*navigation list*) übernommen, innerhalb dessen `<li>`-Elemente mit `shape` und `coord`-Attributen die Bereichsdefinition übernehmen.

14.7 b

Präsentationselement: Das Element `<b>` bewirkt die Fettauszeichnung (*bold*) der in den Container eingeschlossenen Textinhalte.

14.7.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext	Darf überall in Block- und Inlineelementen auftreten.
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	XHTML mobile profile, iMode XHTML 1.0 **abgeschafft** in HTML 4.0 mobile, iMode XHTML 1.1

Syntaxbeispiel:

```
<b> ... </b>
```

Defaultrendering:
Der Elementinhalt wird fett ausgezeichnet.

CSS-Umsetzung des HTML-Defaults:

```
b   { display:inline; font-weight:bold; }
```

14.7.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	keine
Internet Explorer	*Access-, Edit-Attribute*

14.7.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "true" | "false"` (ab IE 5.5)

`unselectable = "on" | "off"` (IE 5)

14.7.4 Erläuterung zum Element

Der Inlinecontainer `<b>` bewirkt die Darstellung der eingeschlossenen Textinhalte in fetter Schrift. Es handelt sich um ein reines Präsentationselement, das keine Aussage über die logische Bedeutung seines Inhalts macht. Entspricht die Intention einer Hervorhebung »stark betont«, so ist das logische Element `<strong>` vorzuziehen, das ebenfalls eine Fettauszeichnung bewirkt.

Das Element `<b>` besitzt keine Attribute, die auf HTML-Ebene Einfluss auf seine Darstellung nehmen könnten – mittels seines `title`-Attributs können dem Container jedoch Erläuterungen beigegeben werden, die in Webbrowsern als Tooltip erscheinen und auch von Voice-Browsern berücksichtigt werden. Ansonsten können Screenreader auftretende Fettauszeichnungen (einstellungsabhängig) ignorieren oder anzeigen.

```
<p>Ein <b title="Hervorhebung">fett ausgezeichnetes</b> Wort ist
optisch hervorgehoben.</p>
```

Beim Screenrader HAL (**www.yourdolphin.com**) kann beispielsweise die Option »Weight change« angewählt werden, was in etwa zu folgender Umsetzung führten würde:

»Ein **BOLD** fettausgezeichnetes **NORMAL** Wort ist optisch hervorgehoben« **[Pause]**

Die Stärke der Fettauszeichnung, also der eigentliche »Fettegrad«, ist nicht beeinflussbar, etwa (analog zu <big> oder <small>) durch Verschachtelung mehrerer <b>-Container. Entsprechend macht der Einsatz von <b> innerhalb von Überschriften, die von sich aus eine Fettauszeichnung hervorrufen, keinen Sinn (beides ist jedoch erlaubt).

Die erlaubte Verschachtelung von <b>-Containern kann per CSS für eine zusätzliche Hervorhebung durch die Schriftfarbe eingesetzt werden (wir befinden uns ja ohnehin bereits auf der Präsentationsebene):

```
b  b  {color:#f00; }   /* bold in bold rot darstellen */
```

Mittels CSS kann die Wirkung von <b> auch aufgehoben oder abgestuft werden. Letzteres muss jedoch unterstützt werden, was im Browser in der Regel daran scheitert, dass nicht genügend abgestufte Schriftschnitte vorhanden sind:

```
<p>Ein <b style="font-weight:normal">fettausgezeichnetes</b> Wort
ist optisch hervorgehoben.</p>
```

Folgende Anweisung hätte – in der Theorie – eine halbfette Auszeichnung des eingeschlossenen Wortes zur Folge:

```
<style type="txt/css">
b.halbfett { font-weight:500; }
</style>

<p>Dieses Wort wird <b class="halbfett">halbfett</b>
dargestellt.</p>
```

In der Praxis stehen dem leider zwei Aspekte entgegen – zum einen stehen dem Browser meist nur zwei Schriftschnitte (»normal« und »fett«) zur Verfügung, so dass nur das eine oder andere Extrem darstellbar ist (bei numerischen Stärke-Angaben werden Werte von 600 und darüber gewöhnlich fett ausgezeichnet, alle darunter liegenden normal). Für reine Bildschirmdarstellung macht abgestufte Fettauszeichnung daher wenig Sinn – anders ist die Situation bei Printstylesheets für Druckausgabe, da hier gegebenenfalls mehr Schriftschnitte zur Verfügung stehen.

Das zweite Problem ist, dass das HTML-Defaultrendering einem Inlinestyle gleichrangig gewertet wird, eine (als Beispiel) Zuweisung »normal« per Styleklasse also der durch das Element selbst veranlassten Präsentation »bold« unterliegt – Folge: Die Darstellung bleibt fett. Abhilfe schafft nur die direkte Zuweisung per Inlinestyle.

Anmerkung: In XHTML 2.0 ist das Element <b> abgeschafft, da es rein präsentativen Zwecken dient. Hierfür sollen ausschließlich Stylesheetformatierungen herangezogen werden.

14.8 base

Das Element `<base>` legt das Basis-Target aller im Dokument auftretenden Hyperlinks sowie den Basis-URL aller relativen URLs fest.

14.8.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Headkontext
XHTML-Modul	Base (zusätzl. sinnvoll: Target Modul)
Kontext	Das Element `<base/>` tritt ausschließlich im `<head>`-Container auf.
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	cHTML, XHTML mobile profile iMode XHTML 1.0, 1.1 **abgeschafft** in HTML 4.0 mobile

Syntaxbeispiel:
`<base ...>`(HTML)

`<base ... />` (XHTML)

Defaultrendering:
Das Element wird (als Inhalt des `<head>`-Containers) nicht dargestellt.

CSS-Umsetzung des HTML-Defaults:

```
/* display-Property geerbt von head (inheritance): */
base   { display:none; }
```

14.8.2 Attribute (Übersicht)

Standardattribute

HTML	–
XHTML	–

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onreadystatechange`

Elementspezifische Attribute

W3C	`href`, `target`
Internet Explorer	`id`

14.8.3 Offizielle Attribute – W3C

href

Wert	*URL*
Typ	Ein absoluter oder relativer URI-String
Verwendung	Optional

Das `href`-Attribut von `<base/>` bezeichnet mit einem absoluten URI-String den absoluten Basis-URI, auf den sich relative URIs im Dokument beziehen. Durch die Übergabe eines relativen URI-Strings ergibt sich der Bezugs-Basis-URI aus ersterem und dem realen Basis-URI[12] des Dokuments (*document URL*).

target

Wert	*nmtoken*
Typ	Freigewählter oder festgelegter Bezeichner
Verwendung	Optional

(XHTML Mod: Target) Ein freigewählter Bezeichner eines (existierenden) Browserfensterobjekts oder – im Framesetkontext – der Bezeichner eines benannten Frameobjekts bzw. einer der vordefinierten Bezeichner `"_self"`, `"_parent"`, `"_top"` oder `"_blank"`.

12 Die Spezifikation weist darauf hin, dass dies nicht notwendigerweise dem Dokument-URI entspricht, sondern stets der realen, nicht durch HTTP-Umleitungen beeinflussten Dokumentadresse.

Ein Hyperlink ohne eigenes `target`-Attribut wird in dem über das `target`-Attribut des `<base>`-Elements bezeichneten Fenster oder Frame ausgeführt:

- `_self`:
  Der Link wird im gleichen Fensterobjekt des aktuellen Framekontexts ausgeführt.
- `_parent`:
  Der Link wird im unmittelbar dem aktuellen Frame übergeordneten Frameobjekt des aktuellen Framekontexts ausgeführt.
- `_top`:
  Der Link wird im hierarchisch obersten Frame des aktuellen Framekontexts ausgeführt (auch als »Hauptfenster« bezeichnet; der Frameset wird entladen).
- `_blank`:
  Der Link wird in einem sich neu außerhalb des aktuellen Framekontexts öffnenden Fenster ausgeführt.
- Ein frei gewählter Bezeichner
  führt den Link im gleichnamigen Fensterobjekt aus (falls vorhanden – dies kann ein benanntes Frameobjekt oder ein frei schwebendes Fenster sein) oder öffnet ein Browserfenster, das anschließend unter diesem Bezeichner ansprechbar ist.

[»] *Hinweis:* Der vordefinierte Bezeichner `_blank` öffnet ebenfalls ein neues Fenster, vergibt an dieses aber **keinen Bezeichner**, so dass dieses Fenster in Folge über Links nicht erneut ansprechbar ist.

Globales Target unterliegt lokalem Target
Ein explizit pro Link gesetztes Target hat Vorrang vor einem mit Hilfe des `<base>`-Elements und dessen `target`-Attribut gesetzten globalen Targets.

14.8.4 Proprietäre Attribute – Internet Explorer

id

Wert	*ID*
Typ	Zeichenkette in Form eines Identifiers
Verwendung	Optional

Der Internet Explorer unterstützt für das Element `<base>` auch das `id`-Attribut. Dies ermöglicht, das `BaseHref`-Property des Elements per Scripting auszulesen.

14.8.5 Erläuterung zum Element

Das <base>-Element dient zweierlei Zwecken – erstens definiert es einen Basis-URI für das Dokument, in dessen <head>-Container es eingesetzt ist, zweitens kann (auch gleichzeitig) ein Base-Target für alle vom Dokument ausgehenden Hyperlinks festgelegt werden.

Setzen eines Basis-URLs

Ein Basis-URI dient als Ausgangspunkt aller anderen, im Dokument verwendeten URIs:

- Hyperlinks (in Ankern und Imagemaps)
- Ressourcen eingebetter Objekte (Bilder, Multimediaobjekte)

Für ein <base>-Element mit href-Attribut der Form

```
<base href="http://www.beispieldomain.de/" />
```

wird beispielsweise der src-URI einer eingebetteten Grafike

```
<img src="illu/beispiel.gif" />
```

aufgelöst zu:

```
<img src="http://www.beispieldomain.de/illu/beispiel.gif" />
```

Ein siteinterner Hyperlink <a href="bereich_a/beispiel.html"> bezieht sich damit stets auf das Wurzelverzeichnis der Domain; die eigentliche Position des Dokuments innerhalb der Site, in dem sich der Link befindet, ist irrelevant.

Basis-URI und Site-Redirect

Sind offizielle Domain und Speicherort einer Site nicht identisch, verfügt man also beispielsweise über einen Account, der lediglich einen Redirect von http://www.meinedomain.de auf einen realen Speicherort, beispielsweise http://members.gratiswebspacebeispiel.com/~meinname/[13] oder vergleichbar zulässt, so werden Suchmaschinen alle auf die Einstiegsseite folgenden Inhalte unter ihrer realen Adresse indizieren (was nicht erwünscht ist).

Eine Lösung hierfür ist, innerhalb einer Site grundsätzlich URIs zu verwenden, die sich auf das Wurzelverzeichnis der Domain beziehen, etwa

```
<a href="/beispiele/bsp_1.html">
```

und gleichzeitig in jedem Dokument einen Basis-URI zu setzen:

13 Dies ist selbstverständlich ein fiktiver URL, wie alle hier in Beispielen verwendeten URLs.

```
<base href="www.beispieldomain.de/" />
```

Durch die Auflösung der URLs durch die Suchmaschine erfolgt die Indizierung wie erwünscht unter dem Domainnamen statt unter der wirklichen Hostadresse.

Das gleiche Prinzip kann eingesetzt werden, wenn die Inhalte einer Site unter mehreren Domainnamen erreichbar ist (z. B. in alternativen Schreibweisen o. ä.), von denen jedoch nur einer als »offizieller« Domainname indiziert werden soll.

Das target-Attribut im Framesetkontext

Anstatt in einem Frameset, beispielsweise in der Navigationsseite, für jeden Hyperlink einzeln ein Target zu setzen, kann hierfür das `<base>`-Element verwendet werden. Als Targetbezeichner wird der geeignete, im Frameset-Dokument vereinbarte Framename verwendet:

```
<!-- im Navigationsdokument: -->
<base target="inhaltframe" />
```

Das target-Attribut ohne Framesetkontext

Für das gelegentlich auftretende Problem, dass eine Website unautorisiert in einem fremden Frameset landet, bietet das `<base>`-Element eine halbwegs passable, simple Lösung frei von JavaScript[14] – es ist lediglich erforderlich, das Base-Target auf das hierachisch oberste Fensterobjekt `_top` zu lenken:

```
<base target="_top">
```

wobei zusätzlich noch ein Basis-URI übergeben werden kann:

```
<base href="http://www.meinedomain.de" target="_top" />
```

Ebenso kann für Dokumente, deren Links überwiegend auf externe Domains zeigen (Linklisten), per Base-Target ein neues Fenster geöffnet werden:

```
<base target="_blank" />
```

oder auch, wenn man nicht in jedem Fall ein neues Fenster wünscht, z. B. mit

```
<base target="extern" />
```

wodurch dem ersten so geöffneten Fenster ein Bezeichner gegeben wird (beliebig wählbar; hier `"extern"`), in dem anschließend sämtliche weiteren Links des Dokuments ausgeführt werden. (Interne, im Hauptfenster auszuführende Links benötigen in diesem Fall allerdings explizit das Target `"_self"`.)

14 Natürlich nicht für die geframte Seite selbst, sondern nur für weitere, von ihr aus verlinkte Seiten der eigenen Domain. Ohne Target blieben diese im Frame.

Seitenblick auf CSS 3

Im Rahmen des »CSS3 Hyperlink Presentation Module« ist für Hyperlinks ein Target-Property geplant[15], dessen Funktionalität die des `<base>`-Elements weit übertrifft. Das `target`-Attribut für Hyperlinks (ebenso das `target`-Attribut des `<base>`-Elements) könnte somit ersatzlos entfallen und durch CSS-Klassenzuweisungen ersetzt werden.

14.9 basefont

Deprecated: Der `<basefont />`-Tag definiert eine Ausgangsschriftgröße, auf die sich die Schriftgrößen der auf den Tag folgenden Textabschnitte (nicht Überschriften) des Dokuments beziehen.

14.9.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Leeres Element
XHTML-Modul	Legacy
Kontext	Der `<basefont>`-Tag darf im Dokumentkopf sowie an beliebiger Position (auch wiederholt) im Dokumentkörper auftreten.
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0 (tr/fs), 4.01 (tr/fs) XHTML 1.0 (ts/fs) **abgeschafft** in HTML 4.0 strict, 4.01 strict XHTML 1.0 strict, XHTML 1.1, XHTML 2.0
Mobil-Spezifikationen	iMode XHTML 1.0 **abgeschafft** in iMode XHTML 1.1, XHTML mobile profile

Syntaxbeispiel:
`<basefont>` (HTML)
`<basefont />` (XHTML)

Defaultrendering:
Das Element selbst wird nicht angezeigt, sondern bestimmt lediglich die Darstellung der folgenden Inhalte.

15 Siehe http://www.w3.org/TR/css3-hyperlinks/.

CSS-Umsetzung des HTML-Defaults:

```
basefont { display:none }
```

14.9.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `title`, `lang`, `style`, `id`
XHTML	`id`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onreadystatechange`

Elementspezifische Attribute

W3C	`color`, `face`, `size`

14.9.3 Offizielle Attribute – W3C

color

Wert	*#RRGGBB-Hexangabe* | *farbname*
Typ	Farbwert oder -bezeichner
Verwendung	Optional

Das `color`-Attribut setzt die Schriftfarbe für die auf den `<basefont>`-Tag im Quelltext folgenden Textblöcke und Überschriften. Seine Wirkung erstreckt sich nur bis zum nächsten, folgenden `<basefont>`-Tag, ab welchem die in jenem definierte Farbe gilt, oder aber, sofern dort kein color-Attribut gesetzt ist, die Defaultschriftfarbe des Dokuments.

[!] *Achtung:* Schriftfarbe von Texten in Tabellenzellen (unmittelbar oder in `<p>`-Containern) wird nur durch einen im Dokumentkopf stehenden `<basefont>`-Tag beeinflusst (ansonsten wird die Defaultfarbe bzw. die im `<body>` vereinbarte Farbe verwendet).

face

Wert	*fontname_1 [, fontname_2, ... fontname_N]*
Typ	Kommagetrennte Liste aus einem bis N Fontbezeichnern.
Verwendung	Optional

Das Attribut `face` nennt den für die folgenden Textabschnitte und Überschriften zu wählenden Font und, in der übergebenen Liste auf diesen folgend, alternative Fonts, auf die der Reihenfolge nach zurückgegriffen werden soll. Es ist sinnvoll, hier Schriften gleicher Gattung anzugeben und von der exakten Forderung über die Alternativen zur allgemeinen Gattung abzustufen, etwa:

```
face = "arial, helvetica, sans-serif"
```

Die Gültigkeit des `face`-Attributs erstreckt sich nur bis zum nächsten folgenden `<basefont>`-Tag und auch dann nicht über diesen hinaus, wenn in ihm selbst kein `face`-Attribut gesetzt ist. In diesem Fall wird auf die Browser-Defaultschrifteinstellung zurückgegriffen. Das `face`-Attribut wirkt ebenfalls auf Überschriften und in Tabellen eingeschlossenen Text.

size

Wert	*n* | *+n* | *-n*
Typ	Numerisch 1 bis 7; ggfs. mit Vorzeichen + oder –
Verwendung	Optional

Setzt die Schriftgröße für die folgenden Textabschnitte. Die Gültigkeit der durch das `size`-Attribut vorgenommenen Schriftgrößenangabe endet beim nächsten folgenden `<basefont>`-Tag, ab dem entweder dessen Angabe gilt, oder, falls dieser kein `size`-Attribut besitzt, die Defaultschriftgröße 3.

Die Größe kann entweder in den absoluten Größenangaben 1 (sehr klein) bis 7 (sehr groß) erfolgen oder in relativen Angaben +1 bis +7 (größer) bzw. –1 bis –7 (kleiner), die sich auf die Ausgangsschriftgröße 3 beziehen (hierbei können jedoch in der Summe mit eventuellen Größenangaben durch `<font>`-Container die Extremwerte Größe 7 und Größe 1 nicht über- bzw. unterschritten werden).

Achtung: Überschriften sind durch das `size`-Attribut nicht betroffen. Schriftgröße von Text in Tabellenzellen (egal ob unmittelbar oder in `<p>`-Containern) wird nur durch den `<basefont>`-Tag nicht beeinflusst, auch wenn dieser im Dokumentkopf steht (es wird stets die Defaultgröße verwendet). **[!]**

14.9.4 Erläuterung zum Element

Historisches Element – vom Einsatz wird stark abgeraten!

Die Verwendung von `<basefont>` ist nur in Zusammenhang mit der gleichzeitigen Nutzung des `<font>`-Containers sinnvoll, von der allerdings ebenfalls nur abgeraten werden kann. Problematisch sind zusätzlich die uneinheitliche, stark kontextabhängige Wirkung der `<basefont>`-Attribute auf folgende Textabschnitte und Überschriften – so wirken auf Überschriften lediglich das `face`- und das `color`-Attribut, und `<basefont>` wirkt ebenfalls nicht in Tabellen hinein (es sei denn, man platziert `<basefont>`-Elemente in Überschriftencontainer oder Tabellenzellen, was mit der Wirkung ebendort eingesetzter `<font>`-Tags vergleichbar ist).

Der `<basefont>`-Tag ist nur Teil der *transitional* bzw. *frameset*-DTDs von HTML 4.01 und XHTML 1.0. In den *strict*-DTDs und den Folgeversionen XHTML 1.1 und 2.0 ist er nicht enthalten.

Achtung – Uneinheitliche Unterstützung

Einige Browser haben die Unterstützung von `<basefont>` bereits eingestellt – in Opera, Mozilla, Safari, Konqueror und Firefox hat das Element keine Wirkung mehr. Internet Explorer dagegen führt `<basefont>`-Anweisungen derzeit (noch) weiterhin aus.

14.10 bdo

Der `<bdo>`-Container (*bidirectional override*) definiert, mit Vorrang vor der Defaultrichtung (*base directionality*), die Schreibrichtung der von ihm eingeschlossenen Inhalte.

14.10.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Bidirectional Text
Kontext	Der `<bdo>`-Container darf in allen Block- und Inlineelementen auftreten.
Erlaubte Inhalte	Text und Inlineelemente.
Offizielle W3C Spezifikationen	HTML 4.0, 4.01, XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0

Mobil-Spezifikationen	HTML 4.0 mobile, iMode XHTML 1.0 **nicht enthalten** in XHTML mobile profile, iMode XHTML 1.1

Syntaxbeispiel:

```
<bdo dir="ltr"> ... </bdo>
```

Defaultrendering:
Der Container setzt die Leserichtung anhand seines `dir`-Attributs, bewirkt ansonsten jedoch keine Änderung der Präsentation.

CSS-Umsetzung des HTML-Defaults:

```
bdo[dir="rtl"] { display:inline; direction: rtl }
bdo[dir="ltr"] { display:inline; direction: ltr }
```

14.10.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `title`
XHTML	`class`, `id`, `xml:lang`, `title`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncellchange`, `onclick`, `oncontextmenu`, `oncopy`, `oncut`, `ondblclick`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onerrorupdate`, `onfilterchange`, `onfocus`, `onhelp`, `onkeydown`, `onkeypress`, `onkeyup`, `onlosecapture`, `onmousedown`, `onmousemove`, `onmouseout`, `onmouseover`, `onmouseup`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onscroll`, `onselectstart`

Elementspezifische Attribute

W3C	**`dir`**
Internet Explorer	*Access-, Edit-Attribute*

14.10.3 Offizielle Attribute – W3C

dir

Wert	*"ltr"* | *"rtl"*
Typ	festgelegte Bezeichner
Verwendung	**Obligatorisch**

Erlaubte Werte sind `dir="rtl"` (right-to-left) und `dir="ltr"` (left-to-right). Sie stellen eine Angabe über die Leserichtung der Inhalte des Containers dar, die Einfluss auf die Darstellung im Browser nimmt.

14.10.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "true" | "false"` (IE 5.5)

`unselectable = "on" | "off"` (IE 5)

14.10.5 Erläuterung zum Element

Der `<bdo>`-Container ermöglicht, die Darstellungsrichtung der in ihm eingeschlossenen Textinhalte zu definieren und bei Bedarf umzukehren. Die durch dessen obligatorisches `dir`-Attribut angegebene Richtung des `<bdo>`-Containers hat dabei stets Vorrang vor der »natürlichen« Schreibrichtung.

So ergibt (streng genommen eine Spielerei)

```
<p>Vorwärts</p>
<p><bdo dir="rtl">Rückwärts</bdo></p>
```

in der Browserdarstellung

Vorwärts

sträwkcüR

Zu beachten ist, dass hier wirklich nur die *Präsentation* beeinflusst wird – sobald man den umgekehrten Text aus dem Browserfenster zu kopieren versucht, wird dies deutlich: Die Kopie besitzt nach wie vor die Quelltextrichtung.

Möglich ist die gleichzeitige Verwendung des `lang`-Attributs[16], um die Sprache der darzustellenden Inhalte zu spezifizieren:

```
<p>Das hier ist Hebräisch: <bdo lang="he" dir="rtl">
Hebräisch schreibt man von rechts nach links</bdo>.</p>
```

Ergibt im Browserfenster:

Das hier ist Hebräisch: sknil hcan sthcer nov nam tbierhcs hcsiärbeH.

Als Nebenbemerkung: Grundsätzlich soll der User Agent aus dem Wert eines `lang`-Attributs *nicht* auf die Schreibrichtung der eingeschlossenen Inhalte schließen – auch wenn wie in diesem Fall diese sich hieraus entnehmen ließe (der verwendete Mechanismus wird weiter unten beschrieben).

Bei Umkehrung der Schreibrichtung werden **Interpunktion** und **Leerzeichen** nicht automatisch sinnvoll angepasst, sondern gleichrangig mit den Buchstaben umsortiert:

```
<p>Drei Worte Arabisch (ebenfalls von rechts nach links):
<bdo lang="ar" dir="rtl">eins, zwei, drei</bdo>.</p>
```

ergibt im Browserfenster (man beachte die Stellung der Kommata):

Drei Worte Arabisch (ebenfalls von rechts nach links): ierd ,iewz ,snie.

Das W3C sieht die **Steuerung der Interpunktionsanzeige** durch Entities vor (ihre Unterstützung lässt noch auf sich warten), die jeweils *hinter* das Interpunktionszeichen gesetzt werden und zwar `&rtl;` für Textrichtung von rechts nach links bzw. `&ltr;` für Textrichtung links nach rechts.

Das obige Beispiel wäre demnach so zu erweitern:

```
<p>Drei Worte Arabisch:
<bdo lang="ar" dir="rtl">eins,&rlm; zwei,&rlm; drei</bdo>.</p>
```

mit dem beabsichtigten Effekt

Drei Worte Arabisch: ierd, iewz, snie.

16 Sprachbezeichner gemäß ISO 639; siehe: www.oasis-open.org/cover/iso639a.html.

Wozu das alles?

Die Gründe für die Existenz eines Elements wie `<bdo>` sind nicht trivial. Alles leitet sich ab vom so genannten **Bi-direktionalen Algorithmus**, der die Interpretation von Unicode-Text durch den Browser beschreibt.

Man muss hier unterscheiden zwischen der Grob- und der Feinstruktur eines Dokuments, wobei die Grobstruktur durch die Blockelemente gegeben ist, aus denen sich das Dokument zusammensetzt, die Feinstruktur durch den enthaltenen Text und etwaige Inlineelemente.

Mittels des `dir`-Attributs, beispielsweise im `<html>`-Element, kann eine **Basistextrichtung** (*base directionality*) des Dokuments festgelegt werden. Diese betrifft nur die Grobstruktur, kehrt also nicht, wie man erwarten mag, zwangsläufig die Buchstabenfolge des Textes um, wenn man sie auf »rtl« setzt.

Beeinflusst wird zunächst lediglich die **Defaultausrichtung** der Inhalte der **Blockelemente**. So wird Text in Absätzen mit `dir="rtl"` grundsätzlich rechts ausgerichtet. Achtung – die Position der Interpunktion am Satzende kehrt sich dabei um: ein Punkt rückt für »rtl« auf die linke Seite der Inhalte:

.Das sieht so aus

Die Wirkung des `dir`-Attributs wird dabei nur von Blockelementen an innerhalb dieser liegende Blockelemente vererbt (stellen Sie sich beispielsweise eine verschachtelte `<div>`-Struktur vor). Es geschieht auch keine Umsortierung von Inlineelementen innerhalb eines Blocks.

Die eigentliche Entscheidung über die Schriftrichtung fällt jedoch erst auf der Ebene des Feinstruktur, also des Textes: Aus den **Unicode-Charactercodes** der Zeichen wird erschlossen[17], mit welcher Textrichtung ein Wort bzw. ein Textabschnitt dargestellt werden soll.

Tauchen beispielsweise hebräische Schriftzeichen, die von rechts nach links zu lesen sind, in einem deutschen Textabschnitt auf, so geschieht Folgendes:

```
<p>Gern geschehen heißt איך בערמה auf Hebräisch.</p>
```

wird dargestellt (vergleichen Sie die Folge der Zeichen!) als

Gern geschehen heißt איך בערמה auf Hebräisch.

Beachten Sie, dass es sich beim hebräischen Abschnitt um drei Worte handelt. Dieser Abschnitt wird also in seiner Gesamtheit (und nicht etwa Wort für Wort)

17 Der Algorithmus zur Erkennung bi-direktionalen Texts ist recht komplex – er wird beschrieben unter: www.unicode.org/reports/tr9/.

in der dargestellten Zeichenfolge **umgekehrt**. Die Schreibrichtung der Textabschnitte ist folglich unabhängig vom `dir`-Attribut des Blockelements (hier ohne dieses Attribut defaultmäßig als »ltr« interpretiert).

Dass der Textinhalt dieses Absatzes wirklich in drei Abschnitte zerfällt, sieht man, sobald man `dir` explizit setzt, wie hier:

```
<p dir="rtl">Gern geschehen heißt המ רעב ריא auf Hebräisch.</p>
```

Wiederum wird nicht die Schreibrichtung innerhalb der drei nach ihrer Textrichtung unterscheidbaren Textabschnitte beeinflusst, wohl aber deren dargestellte Reihenfolge. Der erste deutschsprachige Abschnitt wandert folglich nach rechts (also an die erste Position gemäß der umgekehrten Basistextrichtung!), der Punkt steht wie erwartet links. Der Block wird rechtsbündig angezeigt:

.auf Hebräisch איך בער מה Gern geschehen heißt

Merksatz zum dir-Attribut

Das `dir`-Attribut beeinflusst die Basistextrichtung und damit die Darstellungsreihenfolge von Textbereichen eines Blocks, die sich in ihrer Schreibrichtung unterscheiden – unabhängig davon, ob diese für sich in Inlineelementen stehen oder nicht.

Will man nun eine Zeichenfolge eines Textabschnittes in jedem Fall in einer festgelegten Darstellungsrichtung präsentieren, so muss der Bi-direktionale Algorithmus außer Kraft gesetzt werden – dies geschieht mit Hilfe des `<bdo>`-Elements, das eine bestimmte Darstellungsrichtung aller eingeschlossenen Inhalte *erzwingt*.

Erweitert man den oben gezeigten Quelltext um ein `<bdo>`-Element mit `dir="rtl"`, also folgendermaßen:

```
<p dir="rtl"><bdo dir="rtl">Gern geschehen heißt המ רעב ריא auf
Hebräisch.</bdo></p>
```

… so erfolgt die Präsentation so:

.hcsiärbeH fua איך בער מה tßieh nehehcseg nreG

Zu beachten ist, dass sich die Schreibrichtung der hebräischen Inhalte gegenüber dem oberen Beispiel *nicht* geändert hat – dieser erfolgte vorher schon rechts nach links –, sondern dass nur die deutschsprachigen Inhalte betroffen sind. Deren durch den Bidirektionalen Algorithmus bestimmte Darstellungsrichtung wurde hier widerrufen.

Der umgekehrte Fall wäre, hebräische Zeichen in Quelltextreihenfolge darstellen zu müssen, wobei der Algorithmus entsprechend zu überlisten wäre. Nehmen wir folgenden Sachverhalt an:

```
<p>Das hebräische Wort תוליעפ setzt sich (von rechts nach links
gelesen) aus der Buchstabenfolge פ, ע, י, ל, ו, ת zusammen.</p>
```

… was der Browser stur mit

> Das hebräische Wort פעילות setzt sich (von rechts nach links gelesen) aus der Buchstabenfolge ת, ו, ל, י, ע, פ zusammen.

wiedergibt. Dies liegt daran, dass die hebräischen Buchstaben hier zwar einzeln stehen, aber nur durch Kommata und Leerzeichen getrennt sind – diese jedoch gelten als richtungsmäßig »neutral« (*directionally neutral*). Die Folge davon ist, dass die gesamte Passage als »hebräisch« erkannt und richtungsmäßig umgekehrt wird. Erst wenn die Buchstabenfolge in ein `<bdo>`-Element eingeschlossen ist, lässt sich die gewünschte Darstellung erzielen:

```
<p>Das hebräische Wort תוליעפ setzt sich (von rechts nach links
gelesen) aus der Buchstabenfolge <bdo dir="ltr"> פ, ע, י, ל, ו, ת
</bdo> zusammen.</p>
```

… ergibt, wie beabsichtigt:

> Das hebräische Wort פעילות setzt sich (von rechts nach links gelesen) aus der Buchstabenfolge פ, ע, י, ל, ו, ת zusammen.

Nicht anwendbar über mehrere Textabsätze hinweg:

Nicht möglich ist der Einsatz von `<bdo>` außerhalb von Blockelementen, beispielsweise in der Absicht, die Textrichtung über mehrere Absätze zu ändern (das folgende Beispiel »tut« zwar etwas, wenn auch nicht das Erwartete):

```
<bdo dir="rtl">
<p>Die Textrichtung hier ändern.</p>
<p>... und auch in diesem Absatz.</p>
</bdo>
```

Tatsächlich wird hier lediglich die Basistextrichtung der Absätze geändert (Rechtsausrichtung), die Wirkung entspricht also der eines jeweils direkt oder für ein umliegendes Blockelement gesetzten `dir`-Attributs:

```
<p dir="rtl">Die Textrichtung hier ändern.</p>
<p dir="rtl">... und auch in diesem Absatz.</p>
```

bzw.

```
<div dir="rtl">
<p>Die Textrichtung hier ändern.</p>
<p>... und auch in diesem Absatz.</p>
</div>
```

Für die beabsichtigte Wirkung müsste ein <bdo>-Element innerhalb jeden Blokkelements gesetzt werden (dies wäre allerdings links ausgerichtet):

```
<p><bdo dir="rtl">Die Textrichtung hier ändern.</bdo></p>
<p><bdo dir="rtl">... und auch in diesem Absatz.</bdo></p>
```

Um die Sache konsequent zu Ende zu bringen (Schreibrichtung und Ausrichtung) müsste zusätzlich von außen (oder wieder pro Absatz) die Basistextrichtung geändert werden:

```
<div dir="rtl">
<p><bdo dir="rtl">Die Textrichtung hier ändern.</bdo></p>
<p><bdo dir="rtl">... und auch in diesem Absatz.</bdo></p>
</div>
```

Dies könnte auch dokumentweit im <html>-Element vorgenommen werden:

```
<html dir="rtl">
...
<p><bdo dir="rtl">Die Textrichtung hier ändern.</bdo></p>
<p><bdo dir="rtl">... und auch in diesem Absatz.</bdo></p>
...
</html>
```

Anmerkung: In XHTML 2.0 *entfällt* das <bdo>-Element, da die erlaubten Werte des dir-Standardattributs erweitert werden. Erlaubt sind zusätzlich zu "ltr" (left-to-right, Default) und "rtl" (right-to-left) die Werte "rto" (right-to-left override) und "lto" (left-to-right override), was es ermöglicht, für jeden beliebigen Elementcontainer einen *bidirectional override* zu definieren.

14.11 big

Präsentationselement: Das Element <big> dient dazu, den eingeschlossenen Text auf ca. 120 % Schriftgröße zu vergrößern – er kann zur Erhöhung der Wirkung verschachtelt werden.

14.11.1 Elementeigenschaften

Einführung	HTML 3.2 (konzipiert in HTML 3.0)
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext	Das Element <big> ist überall in Blockelementen erlaubt.

Erlaubte Inhalte	Text und Inlinelemente, darunter auch `<big>` selbst. Das Element darf dabei beliebig verschachtelt werden (hier ist die Wirkung uneinheitlich).
Offizielle W3C Spezifikationen	HTML 2. 0, 3.2, 4.0, 4.01, XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	XHTML mobile profile, iMode XHTML 1.0 **abgeschafft** in HTML 4.0 mobile, iMode XHTML 1.1

Syntaxbeispiel:

```
<big> ... </big>
```

Defaultrendering:

Der Containerinhalt wird mit ca. 120 % vergrößerter Schrift dargestellt.

CSS-Umsetzung des HTML-Defaults:

```
big         { font-size: 1.2em }
```

14.11.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.11.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.11.4 Erläuterung zum Element

Das Element `<big>` bietet eine simple Möglichkeit, ohne den Einsatz von CSS eingeschlossene Textpassagen gegenüber dem umliegenden Text leicht vergrößert darzustellen – die Zeilenhöhe wächst hierdurch entsprechend, was beachtet werden sollte. Der Vergrößerungsfaktor beträgt ca. 1,2. Der Tag ist überall in Blockelementen einsetzbar.

Bei einer Verschachtelung mehrerer `<big>`-Container kaskadiert sich deren Wirkung – der Vergrößerungsfaktor des innenliegenden Containers bezieht sich in diesem Fall auf den bereits vergrößerten Inhalt des äußeren Containers.

So könnte sich durch mehrfache Verschachtelung ein Initial erzeugen lassen:

```
<p><big><big><big>S</big></big></big>uspendisse quis ante.
 Proin sem nisl, ... </p>
```

Was im Browser etwa folgendermaßen aussähe:

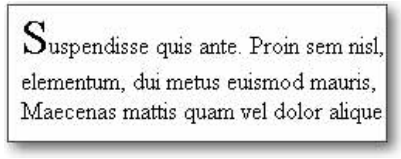

Abbildung 14.8 Initial mit verschachtelten <big>-Elementen

In der Praxis wird man für ein Initial heutzutage wegen der (wesentlich besser steuerbaren) Wirkung den CSS-Pseudo-Selektor `first-letter` verwenden:

```
p.initial:first-letter { font-size:1.8em; }  /* ca. 3 x big */
<p class="initial">Suspendisse quis ante.
 Proin sem nisl, ... </p>
```

Die obere Grenze einer mit `<big>` möglichen Vergrößerung ist abhängig vom Browser bzw. dessen Implementierung des Elements – so tritt beim Internet Explorer bei normaler Basisschriftgröße ab fünffacher Verschachtelung von `<big>`-Elementen keine weitere Vergrößerung mehr auf (hier entspricht die Implementierung dem `size`-Attribut von `<font>`; bei `size="+5"` ist daher Schluss). In Überschriften erster Ordnung (`<h1>`) ist daher im IE eine Verschachtelung wirkungslos (die maximale Schriftgröße 7 ergibt sich bereits durch das erste eingesetzte `<big>`; bei kleineren Überschriften ist entsprechend mehr »Headroom«.).

Andere Browser (Firefox, Mozilla) scheinen die Vergrößerung auf `em`-Basis vorzunehmen und hier kein Limit zu kennen – dies betrifft auch die Einsetzbarkeit des Elements in Überschriften.

Statt `<big>` zu verwenden, kann man auch `<span>` mit einer CSS-Styleanweisung einsetzen:

```
span.big    { font-size: 1.2em }  /*  entspricht big      */
span.big2   { font-size: 1.5em }  /*  entspricht 2 x big  */
span.big3   { font-size: 1.8em }    /*  entspricht 3 x big  */
```

... alternativ funktioniert als Substitution für eine `<big>`-Ebene auch der relative `font-size`-Propertywert `larger`.

Zukunft:

In der derzeitigen Draft zu XHTML 2.0 ist das Element `<big>` nicht enthalten.

14.12 blockquote

Das Element `<blockquote>` dient der Kennzeichnung einer längeren Textpassage als Zitat, wurde häufig seiner Präsentation (Einrückung) wegen eingesetzt.

14.12.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	Text

Kontext	Der blockquote-Container darf in folgenden Elementen verwendet werden: <blockquote>, <body>, <button>, , <dd>, <del>, , <fieldset>, <form>, <ins>, , <noscript>, <object>, <td>, <th>.
Erlaubte Inhalte	Beliebige Blockelemente, darunter auch <blockquote> selbst, beliebige Inlineelemente und Text (#PCDATA).
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile iMode XHTML 1.0, 1.1, XHTML mobile profile

Syntaxbeispiel:

```
<blockquote> ... </blockquote>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
blockquote   { display: block;
               margin-top:    1.33em;
               margin-right:  40px;
               margin-bottom: 1.33em;
               margin-left:   40px;
               }
```

14.12.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	`cite`
Internet Explorer	*Access-, Edit-Attribute*

14.12.3 Offizielle Attribute – W3C

cite

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional
Einführung	HTML 4

Das `cite`-Attribut enthält eine Quellenangabe des Zitatblocks in Form eines URL, der auf die zitierte Ressource zeigt.

[!] *Achtung:* Das `cite`-Attribut wird im Internet Explorer nicht unterstützt.

14.12.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.12.5 Erläuterung zum Element

Das Blockelement `<blockquote>` kennzeichnet ein längeres Zitat, das sich auch über mehrere Absätze erstrecken darf. Der `blockquote`-Container kann daher auch selbst Blockelemente wie `p`- und `div`-Container oder auch Tabellen und wiederum andere `blockquote`-Container enthalten. Enthält der `blockquote`-Container unmittelbar Text, so verhält er sich (abgesehen von seiner Präsentation mit Einrückung), was die Abstände zu vorausgehenden und folgenden Blockelementen angeht, analog zum `p`-Container.

Statische Texteinrückung

Die Texteinrückung erfolgt browserabhängig statisch oder variabel. Das `<blockquote>`-Element als Layouthilfe einzusetzen, ist daher riskant (abgesehen davon, dass dies semantisch sowieso falsch ist).

Präsentation

Die übliche Präsentation einer Blockquote ist eingerückt, wobei sich ein Abstand von konstant jeweils 40 px zum linken und rechten Rand eingebürgert hat (IE, Mozilla etc.). Diese statische Einrückung ist unabhängig von der angezeigten Schriftgröße, wird also durch Textzoom oder vergleichbare Einstellungen des Browsers nicht beeinflusst. Manche Browser (beispielsweise Opera) definieren jedoch die Einrückung abweichend auf Grundlage der verwendeten Basisschriftgröße, entsprechend mit etwa 2.5 em – diese hängt dann auch von der durch den User gewählten Schriftgröße (Zoomfaktor) ab.

Die Verschachtelung mehrerer `blockquote`-Container addiert deren seitliche Ränder und verkürzt dementsprechend die zur Verfügung stehende Zeilenbreite.

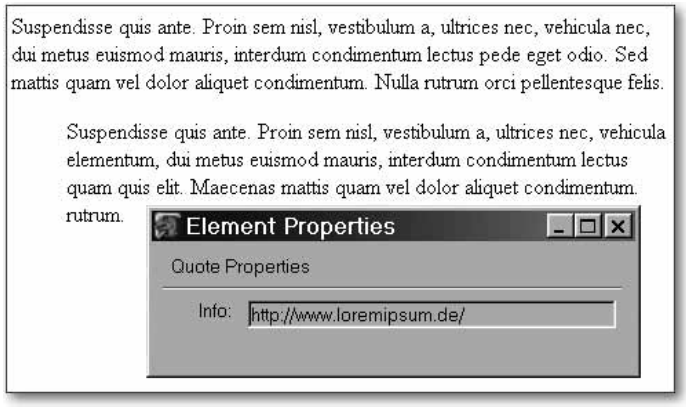

Abbildung 14.9 cite-Attribut einer Blockquote (Kontextmenü, Mozilla)

Quelleninformation mit cite-Attribut

Das `cite`-Attribut dient dazu, einem Zitat eine **Quelleninformation** hinzuzufügen, die von den Browsern allerdings in der Regel nicht direkt zur Verfügung gestellt wird. Mozilla und Firefox sind immerhin in der Lage, die Zitatquelle über den Menüpunkt »Properties« des Kontextmenüs zur Verfügung zu stellen. Eine optische Kennzeichnung einer Blockquote mit beigefügter Quellenangabe erfolgt jedoch nicht, so dass nicht ohne Weiteres kenntlich ist, ob eine solche Angabe vorhanden ist.

Dem ließe sich über CSS »nachhelfen« – eine Unterstützung entsprechender CSS-Selektoren durch den Zielbrowser vorausgesetzt. Denkbar wäre eine farbliche Hervorhebung oder Umrandung des Elements.

Noch besser wäre es, die Zitatquelle selbst darstellen zu können. Die folgende Lösung, das `cite`-Attribut auszuwerten, funktioniert zurzeit nur in Netscape, Mozilla und Firefox:

```
blockquote[cite]:after   {content:'Quelle: ' attr(cite);
                          font-style:italic;
                          display:block; }
```

Die obere CSS-Anweisung setzt den Wert des `cite`-Attributs, falls dieses vorhanden ist, in eine neue Zeile am Ende des Zitatblocks:

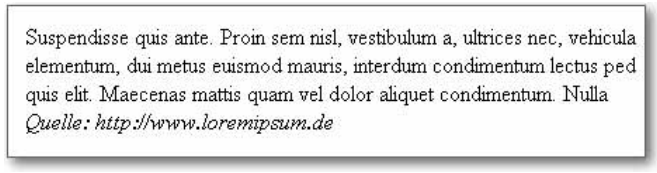

Abbildung 14.10 cite-Attribut einer Blockquote per CSS, dargestellt in Firefox

Hinweis zur Barrierefreiheit

Um **Zitate** korrekt zu interpretieren, benötigt gängige Hilfssoftware die Auszeichnung mit den dafür vorgesehenen Elementen `<blockquote>` oder `<q>`. Nur wenn ein Textabschnitt dementsprechend markiert ist, wird er als Zitat ausgegeben oder verlesen (siehe WAI 3.7). Ungekehrt sollten Sie `<blockquote>` daher nicht als Präsentationselement zur Texteinrückung zweckentfremden.

[zB] *Beispiel:*

```
<blockquote cite="http://www.loremipsum.de">
   Suspendisse quis ante. Proin sem nisl, vestibulum a,
   ultrices nec, vehicula nec, sapien. Vivamus scelerisque
   tempus massa. Sed dictum, quam eu  sagittis elementum,
   dui metus euismod mauris, interdum condimentum lectus
   pede eget odio.
</blockquote>
```

Das `cite`-Attribut nennt den URL einer Quelle, allerdings nicht in der Funktion eines Hyperlinks.

```
<blockquote title="XHTML und CSS, Frank Bongers, 2007">
   XHTML ist nur wenig anders als HTML.
</blockquote>
```

Das `title`-Attribut kann bei Mouseover Informationen über Quelle und Autor des Zitats preisgeben.

14.13 body

Der `<body>`-Container enthält den im Browser sichtbaren Rumpf des HTML-Dokuments. Er gehört als Strukturelement zu den obligatorischen Elementen.

14.13.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Strukturcontainer
XHTML-Modul	Structure
Kontext	Der `<body>`-Container ist Kindelement des Wurzelelements `<html>`.
Erlaubte Inhalte	Alle Blockelemente (Anmerkung: Textinhalt `#PCDATA` unmittelbar im `<body>` ist entgegen früherer Gepflogenheiten nicht mehr gestattet).
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<body> ... </body>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
body    { display: block;
          margin: 8px;
          line-height: 1.33em; }
```

14.13.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	onload, onunload
Internet Explorer Events	onafterprint, onafterupdate, onbeforecut, onbeforepaste, onbeforeprint, onbeforeupdate, onblur, oncontextmenu, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfilterchange, onfocus, onlosecapture, onpaste, onpropertychange, onreadystatechange, onrowenter, onrowexit, onscroll, onselectstart

Elementspezifische Attribute

W3C	alink, background, bgcolor, link, text, vlink
Internet Explorer	bgproperties, bottommargin, leftmargin, nowrap, rightmargin, scroll, topmargin
Netscape	marginheight, marginwidth
Web-TV	fontsize

14.13.3 Offizielle Attribute – W3C

alink

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `alink` bestimmt die Farbe eines aktiven Hyperlinktextes (im Augenblick des Klicks) innerhalb des Dokuments. In Abwesenheit des Attributs gilt die Defaultlinkfarbe für aktive Links `#ff0000` (rot) oder die User-Einstellung des Browsers. Das Attribut ist nicht *case-sensitive*.

```
<body alink="#ff0000"> ... </body>
```

Im *strict*-Vokabular ist das Attribut `alink` nicht enthalten.

CSS-Alternative: `a:active {color:#ff0000;}`

background

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `background` nennt den URL einer Hintergrundgrafik für das Dokument (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich. Übliche Grafikformate sind JPG und GIF, wobei auch transparente und (browserabhängig) animierte GIFs gestattet sind.

```
<body background="bg.gif"> ... </body>
```

Tipp

Eine gleichzeitig mit `bgcolor` definierte Hintergrundfarbe gilt als *hinter* der Hintergrundgrafik befindlich – sie ist durch eventuelle transparente Bereiche einer Hintergrundgrafik sichtbar.

Im *strict*-Vokabular ist das Attribut `background` nicht enthalten.

CSS-Alternative: `body {background-image: url('bg.gif');}`

In CSS ist zusätzlich eine Positionierung und Fixierung der Hintergrundgrafik sowie Angabe einer Richtung der Kachelung möglich.

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `bgcolor` bestimmt die Hintergrundfarbe des Dokuments. In Abwesenheit des Attributs gilt die Defaulthintergrundfarbe `#ffffff` (weiß; früher: `#999999`, grau) oder die User-Einstellung des Browsers. Das Attribut ist nicht *case-sensitive*.

```
<body bgcolor="#ffffff"> ... </body>
```

Im *strict*-Vokabular ist das Attribut `bgcolor` nicht enthalten.

CSS-Alternative: `body {background-color:#fff;}`

link

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `link` bestimmt die Farbe eines nicht-aktiven, noch nicht besuchten Hyperlinktextes innerhalb des Dokuments. In Abwesenheit des Attributs gilt die Defaultfarbe für unbesuchte Links `#0000ff` (blau) oder die User-Einstellung des Browsers. Das Attribut ist nicht *case-sensitive*.

```
<body link="#0000ff"> ... </body>
```

Im *strict*-Vokabular ist das Attribut `link` nicht enthalten.

CSS-Alternative: `a {color:#0000ff;}` oder `a:link {color:#0000ff;}`

text

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `text` bestimmt die Textfarbe innerhalb des Dokuments. In Abwesenheit des Attributs gilt die Defaulttextfarbe `#000000` (schwarz) oder die User-Einstellung des Browsers. Das Attribut ist nicht *case-sensitive*.

```
<body text="#0000ff"> ... </body>
```

Im *strict*-Vokabular ist das Attribut `text` nicht enthalten.

CSS-Alternative: `body {color:#000000;}`

vlink

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `link` bestimmt die Farbe eines besuchten (`visited`) Hyperlinktextes innerhalb des Dokuments. In Abwesenheit des Attributs gilt die Defaultfarbe für besuchte Links `#bb77ff` (lila) oder die User-Einstellung des Browsers. Das Attribut ist nicht *case-sensitive*.

```
<body vlink="#bb77ff"> ... </body>
```

Im *strict*-Vokabular ist das Attribut `vlink` nicht enthalten.

CSS-Alternative: `a:visited {color:#bb77ff;}`

14.13.4 Proprietäre Attribute – Internet Explorer

bgproperties

Wert	*"fixed"*
Typ	fester Bezeichner
Verwendung	Optional
Einführung	IE 3, 4

Das `bgproperties`-Attribut fixiert im Internet Explorer die Position des über `background` angegebenen Hintergrundbildes hinter dem Dokumentkörper (Wasserzeicheneffekt). Einzig erlaubter Wert ist `"fixed"`.

CSS-Alternative: `body {background-attachment:fixed }` bzw. `body {background:url('bg.gif') fixed;}`

bottommargin

Wert	*number*
Typ	Ganzzahliger numerischer Wert größer gleich 0
Verwendung	Optional
Einführung	IE 4

Das `bottommargin`-Attribut legt im Internet Explorer für den **unteren** Dokumentrand den Abstand in Pixel zwischen Inhalten und Fenster- bzw. Framerand fest. Es hat dabei Vorrang vor dem Defaultwert.

CSS-Alternative: `body {margin-bottom: ...px;}`

leftmargin

Wert	*number*
Typ	Ganzzahliger numerischer Wert größer gleich 0
Verwendung	Optional
Einführung	IE 3

Das `leftmargin`-Attribut legt im Internet Explorer für den *linken* Dokumentrand den Abstand in Pixel zwischen Inhalten und Fenster- bzw. Framerand fest. Es hat

dabei Vorrang vor dem Defaultwert. (Das Attribut wurde fälschlich als Synonym zu `marginwidth` von Netscape Navigator verwendet.)

CSS-Alternative: `body {margin-left: ...px;}`

nowrap

Wert	*"yes"* | *"no"*
Typ	festgelegte Werte
Verwendung	Optional
Einführung	IE 4

Das `nowrap`-Attribut mit Wert `"yes"` unterbindet im Internet Explorer den automatischen Zeilenumbruch am Fensterrand. Defaultwert bei Abwesenheit des Attributs ist `"no"`.

CSS-Alternative: `body {white-space: nowrap;}`

Anmerkung: Das Attribut wird an anderer Stelle auch mit den Werten `"true"` und `"false"` verwendet.

rightmargin

Wert	*number*
Typ	Ganzzahliger numerischer Wert größer gleich 0
Verwendung	Optional
Einführung	IE 4

Das `rightmargin`-Attribut legt im Internet Explorer für den *rechten* Dokumentrand den Abstand in Pixel zwischen Inhalten und Fenster- bzw. Framerand fest. Es hat dabei Vorrang vor dem Defaultwert.

CSS-Alternative: `body {margin-right: ...px;}`

scroll

Wert	*"yes"* | *"no"*
Typ	festgelegte Werte
Verwendung	Optional
Einführung	IE 4

Das `scroll`-Attribut erlaubt im Internet Explorer die Anzeige von Scrollbars bzw. unterdrückt diese beim Wert `"no"` auch dann, wenn die Länge der Seite sie erforderlich machen würde. Der Defaultwert ist `"yes"`.

CSS-Alternative: keine

topmargin

Wert	*number*
Typ	Ganzzahliger numerischer Wert größer gleich 0
Verwendung	Optional
Einführung	IE 3

Das `topmargin`-Attribut legt im Internet Explorer für den *oberen* Dokumentrand den Abstand in Pixel zwischen Inhalten und Fenster- bzw. Framerand fest. Es hat dabei Vorrang vor dem Defaultwert. (Das Attribut wurde fälschlich als synonym zu NNs `marginheight` verwendet.)

CSS-Alternative: `body {margin-top: ...px;}`

14.13.5 Proprietäre Attribute – Netscape

marginheight

Wert	*number*
Typ	Ganzzahliger numerischer Wert größer gleich 0
Verwendung	Optional

Das `marginheight`-Attribut legt für den *oberen und unteren* Dokumentrand den Abstand in Pixel zwischen Inhalten und Fenster- bzw. Framerand fest. Es hat dabei Vorrang vor dem Defaultwert. (Das Attribut wurde fälschlich als Synonym zu IEs `topmargin` verwendet.)

CSS-Alternative: `body {margin: ...px 8px;}`

marginwidth

Wert	*number*
Typ	Ganzzahliger numerischer Wert größer gleich 0
Verwendung	Optional

Das `marginwidth`-Attribut legt für den *rechten und linken* Dokumentrand den Abstand in Pixeln zwischen Inhalten und Fenster- bzw. Framerand fest. Es hat dabei Vorrang vor dem Defaultwert. (Das Attribut wurde fälschlich als Synonym zu IEs `leftmargin` verwendet.)

CSS-Alternative: `body {margin: 8px ...px;}`

14.13.6 Proprietäre Attribute – Web-TV

fontsize

Wert	*"small"* | *"medium"* | *"large"*
Typ	festgelegte Bezeichner
Verwendung	Optional
Gültigkeit	nur in Web-TV

Das `fontsize`-Attribut legt die Schriftgröße des Dokuments für den Web-TV-Client unveränderlich fest. Es wird von anderen Browsern nicht unterstützt.

14.13.7 Erläuterung zum Element

Das `<body>`-Element gehört zu den **Strukturelementen**. Es ist als Kindelement des Wurzelelements `<html>` obligatorischer Bestandteil jedes HTML-Dokuments nach *strict*- oder *transitional*-DTD und folgt unmittelbar auf den `<head>`-Container, der den Dokumentkopf bildet. Wird Start- oder Endmarke des `<body>`-Containers in einem HTML-Dokument weggelassen, so wird die fehlende Marke durch den Browser ergänzt. In XHTML dagegen sind sowohl Start- als auch Endmarke vorgeschrieben.

[»] *Hinweis*: Es gehört quasi zum guten Ton, sich auch in Dokumenten gemäß HTML 4.01 nach den strikteren Regeln von XHTML zu richten. Lassen Sie die Elementmarken für `<body>` also keinesfalls weg.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Dokumenttitel</title>
</head>
<body>
   <!-- Dokumentinhalt: -->
   <h1>Beispiel&uuml;berschrift</h1>
   <p>Beispiel f&uuml;r einen Textabsatz.</p>
</body>
</html>
```

Listing 14.1 Grundstruktur eines XHTML-Dokuments

Eine Ausnahme von dieser Regel bilden Framesetdokumente (gemäß der *frameset*-DTD), bei denen an gleicher Position stattdessen das `<frameset>`-Element auftritt. In diesem Fall bildet `<body>` jedoch den Inhalt eines optionalen `<noframes>`-Containers.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Dokumenttitel Frameset</title>
</head>
<frameset>
      <!-- Framedefinitionen -->
  <noframes>
   <body>
      <!-- alternativer Dokumentinhalt -->
   </body>
  </noframes>
</frameset>
</html>
```

Listing 14.2 Grundstruktur eines Frameset-Dokuments (Framedefinitionen weggelassen)

Das `<body>`-Element entspricht dem im Browserfenster sichtbaren Teil der HTML-Seite, also dem Dokumentkörper. Es enthält somit den eigentlichen Inhalt der Webseite.

Zugriff auf <body> mittels DOM

Direkten Zugriff auf den Dokumentkörper haben Sie mittels des DOM-Properties `document.body`, das eine Referenz auf das `<body>`-Element enthält.

Anmerkung: Das Property ist auch auf Frameset-Dokumente anwendbar. In diesem Fall enthält es eine Referenz auf das äußerste `<frameset>`-Element.

Eigenschaften und erlaubte Inhalte des Body-Containers

Beachten Sie, dass Sie unmittelbar im `<body>`-Container nur **Blockelemente** einsetzen dürfen – Inlineelemente oder unformatierter Text (`#PCDATA`) haben hier nichts verloren[18]. Eine Ausnahme bildet der `<script>`-Container sowie jene beiden Inlineelemente, die auch im Blocklevelkontext eingesetzt werden dürfen, nämlich `<ins>` und `<del>`.

Sind für den `<body>` keine Präsentationseigenschaften per CSS oder mittels der entsprechenden HTML-Attribute vereinbart, so treten **Defaulteigenschaften** in Kraft, die durch den Browser oder durch die User-Einstellungen bestimmt sind. Hierzu gehören die Standardfarben für den Dokumenthintergrund sowie für im `<body>` befindliche Textinhalte und Links. Gleichzeitig besitzt der `<body>`-Container ein **Defaultmargin** von 8 Pixel (im Internet Explorer sind es 10 Pixel horizontal und 15 Pixel vertikal).

18 Dies war in der früheren HTML-Version 3.2 noch gestattet.

Wollen Sie die Margins des `<body>` auf Null oder einen anderen Wert setzen (es ist besser, dies explizit zu tun, da die Defaultmargins browserspezifisch verschieden sind), so verwenden Sie hierfür das CSS-Property `margin`.

Anmerkung: Das W3C-Samplestylesheet für HTML-Defaultverhalten, siehe **www.w3.org/TR/REC-CSS2/sample.html**, gibt fälschlicherweise einen *Padding*-wert von 8 Pixel an, doch spricht das Element auf eine entsprechende Nullung des Paddings mit `padding:0` nicht im gewünschten Sinne an.

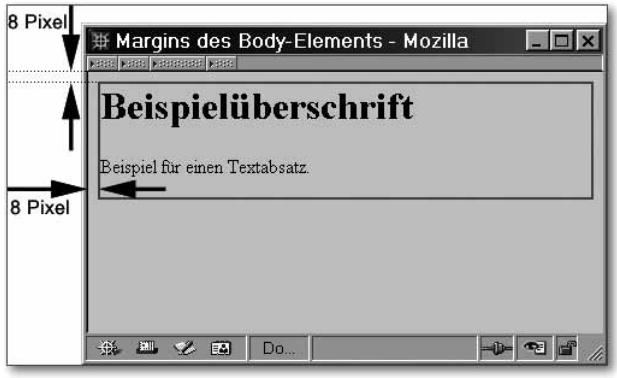

Abbildung 14.11 Defaultmargins des Body (Elementgrenzen hervorgehoben)

Beinahe ein normales Blockelement

Das `<body>`-Element verhält sich weitestgehend wie ein normales Blockelement. Ein Unterschied besteht jedoch darin, dass sich seine Hintergrundfarbe stets auf das gesamte Browserfenster bezieht, also ausdrücklich auch auf die Marginbereiche des Containers. Auch das Hintergrundbild setzt nicht am Rand des Inhaltsbereiches an, sondern am Rand des Browserfensters. Der eigentliche **Inhaltsbereich** des `<body>` wird ansonsten, wie bei anderen Blockelementen auch, nur durch die Abmessungen der in ihm enthaltenen Elementcontainer bestimmt (siehe Abbildung 14.11; Mozilla zeigt die mit durchgezogenem Rahmen markierten Elementgrenzen korrekt an, positioniert die Margins also außerhalb des Rahmens).

Der Internet Explorer bis einschließlich Version 7.0 weicht insofern vom Standardverhalten ab, dass er den Inhaltsbereich des Body prinzipiell auf die gesamte Fläche des Browserfensters ausdehnt und sich hierin auch durch eventuelle Breitenangaben für den `<body>`-Container nicht beirren lässt (siehe Abbildung 14.12; hier ist auch für die Überschrift und den Textabsatz eine Breite und ein Rahmen angegeben).

Abbildung 14.12 Die Body-Margins im IE7 wirken optisch wie Paddings.

Zentrieren des Inhaltsbereiches von <body>

Die Angabe einer Breite für den `<body>`-Container per CSS bietet in standardkonformen Browsern die Möglichkeit, die Seiteninhalte durch Anwendung von `margin:auto` optisch zu zentrieren:

```
body  { width:[breitenangabe]; margin:auto; }
```

Beachten Sie, dass dabei Margins nach oben oder unten automatisch genullt werden (Achtung, auch der Internet Explorer bis Version 6, der `margin:auto` nicht unterstützt, setzt hierbei zwar alle Marginwerte auf Null, beachtet aber weder die Breitenangabe noch die Zentrierungsanweisung).

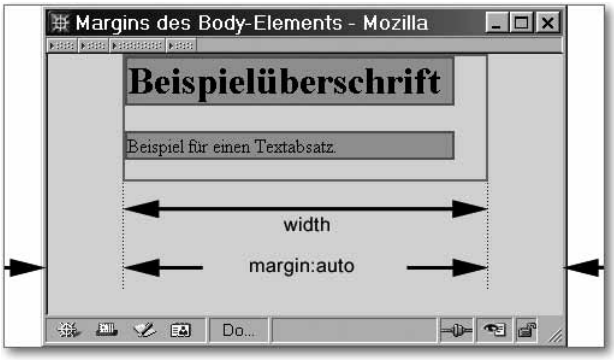

Abbildung 14.13 Zentrieren von Seiteninhalten mit margin:auto

Diese »direkte« Vorgehensweise hat jedoch neben den Problemen mit älteren Versionen des Internet Explorers einige Nachteile – so ist es bei variabler Breite eines durch den `<body>` definierten Inhaltsbereiches problematisch, ein Hintergrundbild gegenüber den Dokumentinhalten zu positionieren.

Zentrieren mit eingeschaltetem -Container

Vorzuziehen daher ist das Einschalten eines Zwischencontainers zum Zweck der Breitenbeschränkung und Zentrierung des Dokumentinhalts. Hierfür eignet sich der neutrale Blockcontainer `<div>`. Die Lösung ist zudem kompatibel zu älteren Browsern. Setzen Sie hierfür jedoch die Body-Margins besser auf Null.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Dokumenttitel</title>
</head>
<body>
   <div id="zentrierer">
      <!-- Dokumentinhalt: -->
      <h1>Beispiel&uuml;berschrift</h1>
      <p>Beispiel f&uuml;r einen Textabsatz.</p>
   </div>
</body>
</html>
```

Listing 14.3 Zwischencontainer im <body>-Element

Mit folgenden CSS-Angaben erreichen Sie eine Zentrierung der Inhalte auch in älteren Internet Explorern bis 6.0:

```
body  {
        margin:0;
        text-align:center;
}

div#zentrierer  {
        width:[breitenangabe];
        margin:auto;
        text-align:left;
}
```

Die Angabe zu `text-align` für den Body bewirkt im Internet Explorer aufgrund einer Fehlinterpretation der CSS-Regeln die Zentrierung aller Elemente innerhalb des `<body>`-Containers, also auch des `<div>`-Zentrierers (zentriert werden dürften nur die Textinhalte, nicht die Blockcontainer selbst). Da so aber auch die Textinhalte weiter innen liegender Container zentriert werden, muss für den `<div>`-Zentrierer die Textausrichtung durch `text-align:left` widerrufen werden (siehe Abbildung 14.4; der zentrierende `<div>`-Container ist durch gestrichelten Rahmen markiert).

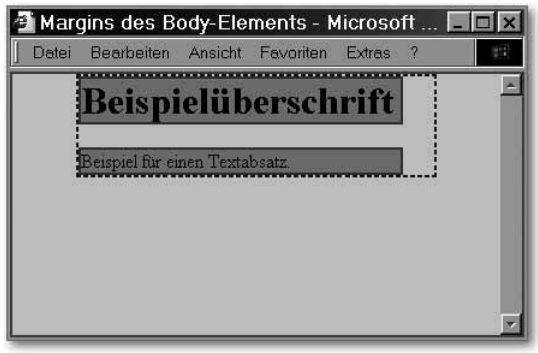

Abbildung 14.14 Zentrierung der Seiteninhalte in IE 6.0

Standardkonforme Browser interpretieren hingegen unmittelbar `margin:auto` des Zentrierers[19]. Der aktuelle Internet Explorer 7.0 unterstützt `margin:auto` allerdings auch direkt, allerdings *nur dann*, wenn Ihr Dokument eine XHTML-Doctype-Deklaration besitzt (seltsamerweise spielt das Vorhanden- oder Nichtvorhandensein eines XHTML-Namensraums in diesem Zusammenhang keine Rolle; dasselbe gilt für einen XML-Prolog).

Internet Explorer 7 benötigt Doctype-Deklaration

Ohne die Doctype-Deklaration schaltet IE7 in den Quirks-Mode und emuliert das Verhalten älterer IE-Versionen inklusive des fehlerhaften CSS-Boxmodells (falsche Abmessungen). Lassen Sie daher die Doctype-Deklaration keinesfalls weg. Für ältere Explorer können Sie zusätzlich den `text-align`-Bug einsetzen, um eine halbwegs identische Darstellung zu erreichen.

14.14 br

Das Element `<br>` (in XHTML `<br/>`) markiert die Position eines erzwungenen Zeilenumbruchs innerhalb eines Blockelements.

14.14.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element
XHTML-Modul	Text

19 Für eine vollständige Angleichung der Optik zwischen den Browsern müssen Sie auch die Margins der innen liegenden Blockcontainer `<h1>` und `<p>` berücksichtigen – wenn Sie hier explizite Werte setzen, sind Sie auf dem richtigen Weg.

Kontext	Das Element ist überall in Blockelementen gestattet.
Erlaubte Inhalte	keine (leeres Element
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

`<br>` (HTML)

`<br/>` (XHTML)

Defaultrendering:

CSS-Umsetzung des HTML-Defaults[20]:

```
br              { display:inline; }
br:before       { content: "\0A"; }
```

14.14.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `title`
XHTML	`class`, `id`, `title`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

W3C	`clear`

20 Der hier als Content eingesetzte Wert `"\0A"` entspricht einem Zeilenumbruchbefehl `LF`. (Dieses Konstrukt wird gegenwärtig nur in Opera unterstützt.)

14.14.3 Offizielle Attribute – W3C

clear

Wert	*"none"* (Default) | *"left"* | *"right"* | *"all"*
Typ	Festgelegte Werte
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `clear`-Attribut ermöglicht den Umbruch in eine neue Textzeile im Kontext eines Float-Objekts (beispielsweise eines Bildes oder einer Tabelle mit `align="left"` oder `align="right"` oder einem entsprechenden CSS-Property). Der Umbruch erfolgt hin zu dem Rand des übergeordneten Blockelements, der durch den Wert des `clear`-Attributs bezeichnet ist.

Der Wert `clear="left"` bricht also gegenüber einem nach links gefloateten Objekt zur nächsten, am linken Rand nach dem Objekt freien Zeile um, `clear="right"` gegenüber einem nach rechts gefloateten Objekt zur nächsten, am rechten Rand freien Zeile.

Der Wert `clear="all"` bricht in Gegenwart links und rechts gefloateter Objekte zur nächsten, an beiden Rändern freien Zeile um. Er verhält sich gegenüber einzelnen gefloateten Objekten entsprechend der beiden anderen Werte.

Der Wert `clear="none"` entspricht dem Elementverhalten bei Abwesenheit des Attributs – er ist lediglich der Vollständigkeit halber deklariert.

CSS-Alternative in strict-Vokabularen obligatorisch

In den *strict*-Vokabularen *muss* anstelle des `clear`-Attributs das CSS-Property *clear* eingesetzt werden; statt `<br clear="all">` also `<br style="clear:both">` und vergleichbar.

14.14.4 Erläuterung zum Element

Das `<br/>`-Element erzwingt einen Zeilenumbruch innerhalb einer Textzeile eines Blockelements, wirkt also wie ein Linefeed-Befehl. Der Zeilenumbruch erfolgt per Definition *vor* dem Tag. Das Element selbst ist nicht sichtbar, verhält sich dabei jedoch wie ein Inlineelement (sein `display`-Property etwa auf `none` zu setzen, unterbindet die Umbruchwirkung).

Der Beginn der neuen Zeile richtet sich nach der Textrichtung (`dir`) des umgebenden Elements bzw. der Textausrichtung (`text-align`).

[zB] *Beispiel:*

```
<p><img align="left" src="bild.jpg" />Bei einem gefloateten
Element, beispielsweise einem Bild, kann ein "clear"-Break
<br clear="left" />zum nächsten freien Rand umbrechen.
In fringilla elementum nunc.</p>
```

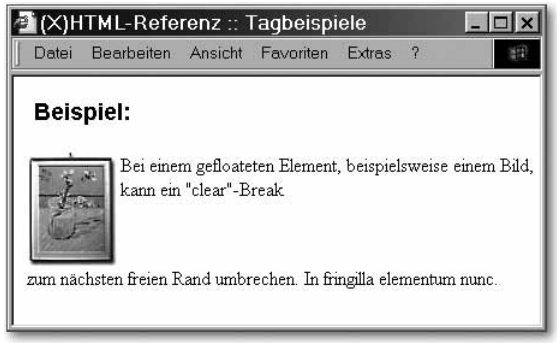

Abbildung 14.15 Ein <br/> mit clear-Attribut beendet das Umfließen eines Bildes.

Probleme mit XHTML-Schreibweise in HTML-Dokumenten

Der Einsatz der in XHTML üblichen »abgeschlossenen« Schreibweise für leere Elemente, hier also `<br/>`, ist in HTML-Dokumenten nicht nur *falsch*, sondern kann obendrein zu unerwarteten Darstellungsproblemen führen.

Der Grund hierfür liegt in der in SGML erlaubten »Shorttag«-Syntax, die den Abschluss eines Containers mit einfachem Slash gestattet. So kann für

```
<p>Beispiel Textabsatz</p>
```

in SGML verkürzend geschrieben werden (auch die Startmarke wird verkürzt):

```
<p/Beispiel Textabsatz/
```

Ein SGML-Validator (und ein HTML-Dokument *ist* ein SGML-Dokumenttyp!) betrachtet dies daher nicht als Syntaxfehler. Einem SGML-konform arbeitendem Browser, der im Rahmen eines HTML-Dokuments auf ein XHTML-Konstrukt wie `<br/>` stößt, könnte folgende Fehlinterpretation unterlaufen:

`<br/>` wird interpretiert als `<br>>`

Dies geschieht, weil `<br/` als Shorttag für den vollständigen Ausdruck `<br>` angesehen wird. Der Browser erwartet anschließend kein weiteres Symbol für den Tagabschluss mehr und betrachtet die folgende schließende Tagklammer als regulären Textinhalt.

Auch das für ältere Netscape-Browser eingebürgerte Einschieben eines Leerzeichens ändert an dieser Situation nicht – teilweise wurden Schreibweisen wie `<br />` auch als zwei aufeinanderfolgende Breaks fehlinterpretiert.

Kein »Beinahe«-XHTML

Beachten Sie, dass es keine Dokumente gibt, die »beinahe« XHTML darstellen! Ein HTML-Dokument wird nicht »besser«, wenn Sie Teile davon gemäß XHTML-Syntax schreiben, es ist lediglich nicht mehr validierbar. Entscheiden Sie sich im zweifelsfall für XHTML, tun Sie dies aber konsequent.

Beliebter Missbrauch

Das `<br>`-Element wird häufig zweckentfremdet, um absatzähnliche Abstände zwischen zwei präsentierten Textblöcken zu erzielen oder einen Abstand nach oder vor einem Blockelement zu vergrößern (die folgenden Beispiele sind allesamt fragwürdiges HTML):

- **Pseudo-Absatz**

```
<p>Dies ist ein Textabschnitt,
der wie ein Absatz aussieht.
<br><br>
Dieser Textabschnitt wirkt
wie ein zweiter Absatz, ist aber keiner.</p>
```

- **Pseudo-Margins**

```
<p><br>
Um mehr Raum vor diesem Absatz zu bekommen, wurde dem Text ein
Zeilenumbruchbefehl vorangestellt.</p>
```

  Zwischen zwei Listenelementen soll etwas mehr Platz gemacht werden:

```
<ul>
<li>Listenpunkt 1 <br> </li>
<li>Listenpunkt 2 <br> </li>
</ul>
```

- **Pseudo-Liste**

  Eine listenähnliche Optik wird hier mit Zeilenumbrüchen und Spiegelstrichen imitiert:

```
<p>Für leckere Eierkuchen reichen:<br/><br/>
- Milch <br/>
- Eier <br/>
- Mehl <br/><br/>
Und etwas Salz darf auch sein...</p>
```

In allen oben angeführten Beispielen sollte stattdessen korrektes HTML verwendet werden, also echte Absatzcontainer und Listen. Die gewünschten Präsentationen erzielt man durch Einsatz von CSS.

Anmerkung: Das Element `<br>` *entfällt* in XHTML 2.0 zugunsten des dort neu eingeführten, expliziteren Zeilencontainers `<l>` (*line*).

14.15 button

Der `<button>`-Container deklariert seinen Inhalt als anklickbare Schaltfläche innerhalb eines HTML-Formulars.

14.15.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Forms
Kontext	In Block- und Inlineelementen (außer `<button>` selbst)
Erlaubte Inhalte	Block- und Inlineelemente (mit Ausnahme von allen Formularelementen, `<a>`, `<iframe>` und `<isindex>`) sowie Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 4.0, 4.01, XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile, iMode XHTML 1.0 **abgeschafft** in XHTML mobile profile, iMode XHTML 1.1

Syntaxbeispiel:

```
<button> ... </button>
```

14.15.2 Attribute Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`

Intrinsic Events Collection	`onblur`, `onfocus`
Internet Explorer Events	`onafterupdate`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `onbeforeupdate`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfilterchange`, `onfocusin`, `onfocusout`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onselectstart`

Elementspezifische Attribute

W3C	`accesskey`, `disabled`, `name`, `tabindex`, `type`, `value`
Internet Explorer	*Access-, Databinding-, Edit-Attribute*

14.15.3 Offizielle Attribute – W3C

accesskey

Wert	*character*
Typ	Alphanumerisches Zeichen, nicht *case-sensitive*
Verwendung	Optional

Barrierefreiheit: Das Attribut `accesskey` legt im Sinne der Accessibility ein (einzelnes) alphanumerisches Tastaturzeichen fest, das als Keyboard-Shortcut den Link ausführt – unter Windows bei gleichzeitig gedrückter Alt-Taste[21], unter Mac OS mit Ctrl-Taste.

```
<button accesskey="c" onclick="alert('Button geklickt!')">
Alt + C eingeben um Button auszuwählen</button>
```

Am Button im oberen Beispiel wird mit Alt+C ein Klick-Event erzeugt.

disabled

Wert	*"disabled"*
Typ	Festgelegter Wert
Verwendung	Optional

21 Unter Windows in Opera-Browsern statt mit Alt mit gleichzeitig gedrückter ⇧- und Esc-Taste. Hintergedanke ist, den Tastaturzugang zum Browsermenü per Alt-Taste nicht zu behindern. Dieser bleibt jedoch bei einzelner Betätigung der Alt-Taste auch bei den anderen Browsern unbeschränkt möglich.

Das `disabled`-Attribut deaktiviert die Schaltfläche, die zwar nach wie vor im 3D-Look erscheint, jedoch nicht mehr anklickbar ist. Textelemente auf der Schaltfläche werden ausgegraut.

```
<button disabled="disabled">
   Button mit <b>fettem</b> Text
</button>
```

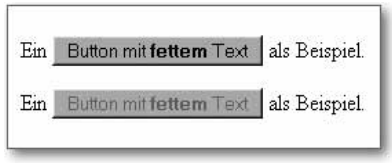

Abbildung 14.16 Buttons anklickbar und disabled (Mozilla/Firefox)

name

Wert	*cdata*
Typ	Textstring als Nametoken-Bezeichner
Verwendung	Optional

Das `name`-Attribut weist einem Button im Formularkontext einen Bezeichner zu, der verwendet wird, wenn für den Button beim Versenden der Formulardaten ein eigener Wert verschickt werden soll.

Anmerkung: Im Formularkontext kann das `id`-Attribut funktional das `name`-Attribut nicht zuverlässig ersetzen, daher bleibt Letzteres nach wie vor zulässig.

tabindex

Wert	*number*
Typ	Ganzzahl
Verwendung	Optional

Barrierefreiheit: Eine Ganzzahl, die in aufsteigender Richtung die Position des Buttons in der Tabulatorwahl-Reihenfolge (⇆-Taste) festlegt.

Die Tab-Reihenfolge ergibt sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes `tabindex`-Attribut in Quelltextreihenfolge.

Die Tab-Reihenfolge enthält stets alle aktivierbaren Elemente eines Dokuments gemeinsam – zu diesen gehören neben `<button>` auch `<a>`, `<area>`, `<input>`, `<object>`, `<select>` und `<textarea>`. Für aktivierbare Elemente sollten daher niemals gleiche Tabindex-Werte gesetzt werden.

Anmerkung: Ein negativer Tabindex entfernt den Button im IE aus der Tab-Reihenfolge, die anderen Browser ordnen ihn in die normale Tab-Reihenfolge ein, als ob kein Tabindex gesetzt wäre.

type

Wert	*"button"* (Default) | *"submit"* | *"reset"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `type`-Attribut steuert das Verhalten des Buttons im Formularkontext. Mit `type="reset"` übernimmt er die Aufgabe eines Reset-Buttons, mit `type="submit"` die eines Submit-Buttons.

Die Abwesenheit des Attributs entspricht dem `type="button"`, somit also dem Defaultverhalten als neutrale Schaltfläche.

```
<button type="submit">Schick <b>mich</b> ab!</button>
<button type="reset">Oder <b>lass</b> es!</button>
```

value

Wert	*cdata*
Typ	Beliebige Zeichenkette
Verwendung	Optional

Das `value`-Attribut weist dem Button einen Wert zu, der zusammen mit dem Wert des `name`-Attributs beim Abschicken des Formulars, das den Button enthält, als *name-value*-Pärchen übertragen wird. Ist ein `name`-Attribut, aber kein `value`-Attribut vergeben, so wird ein *name-value*-Pärchen mit dem leeren String als Wert verschickt. Das Attribut ist *case-sensitive*.

Achtung: Der Internet Explorer verschickt stattdessen, und zwar **auch** (!) bei gesetztem `value`-Attribut, den Textinhalt des Containers. **[!]**

14.15.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (IE 5.5)

IE-Databinding-Attribute

`datafld = cdata` (IE 4)

`dataformatas = "HTML" | "text"` (IE 4)

`datasrc = cdata` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.15.5 Erläuterung zum Element

Der `<button>`-Container kann innerhalb oder außerhalb des Kontextes eines Formulars verwendet werden, um eine fast beliebig gestaltbare Schaltfläche zu erzeugen, mit der beispielsweise Scripte gesteuert werden können. Durch entsprechend gesetztes `type`-Attribut kann der Button in einem Formular auch die Aufgabe eines Reset- oder Submit-Buttons erfüllen.

Der Container darf Text und HTML-Markup enthalten, und zwar sowohl Inline- als auch Blocklevel-Elemente (d.h. alle so genannten Flow-Elemente mit ein paar Ausnahmen). Der Containerinhalt bildet die Oberfläche des Buttons; auch Grafiken können eingesetzt werden.

Mehrzeiliger Inhalt wird auf der Buttonoberfläche per Default zentriert dargestellt, kann mit Hilfe von Blockelementen oder CSS-Anweisungen (`text-align`) jedoch auch beliebig anders ausgerichtet werden.

Inkonsistenzen der Darstellung

Die Darstellung des Buttons bzw. sein Verhalten als Inlinecontainer im Zeilenkontext ist stark browserabhängig (siehe Abbildung 14.17) – so wird der Button in Mozilla an der Zeilenoberkante ausgerichtet, im Internet Explorer an der Zeilenunterkante (was dem Verhalten einer Grafik gleicht). Opera folgt hierin zwar dem Internet Explorer, verursacht jedoch vor dem Button einen Zeilenumbruch, wenn dieser innerhalb eines `<p>`-Containers steht.

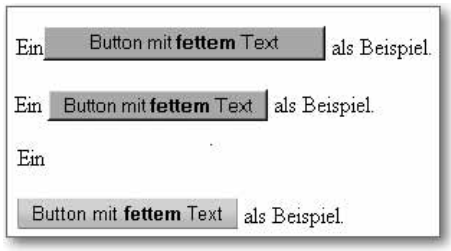

Abbildung 14.17 Buttons in Mozilla (oben), IE und Opera (unten)

Verbotene und sinnlose Inhalte von Buttons

Einige HTML-Elemente sind als Inhalt eines `<button>`-Containers entweder nicht sinnvoll oder nicht gestattet. Gemäß der DTDs von HTML 4.01 und XHTML sind alle Formularelemente (somit auch `<button>` selbst), der Ankertag `<a>` und der `<iframe>`-Tag innerhalb des Containers verboten.

Imagemaps (und Hyperlinks, selbst in jenen Browsern, die den `<a>`-Tag in dieser Position im Gegensatz zur Spezifikation nicht ignorieren) sind als Teil des Buttoninhalts ebenfalls nur schwer nutzbringend einsetzbar. Grund: der Klick-Event am Button selbst tritt stets *vor* dem Klick-Event am inneren aktiven Element (z. B. einem Hyperlink oder einer Area) ein. Dies gilt auch, wenn das innere Element per Accesskey oder Tabulator direkt angesprochen wird. Umgekehrt triggert ein Klick auf den Button in diesen eingebettete Links jedoch nicht.

Als »wertfreie« Demonstration kann folgendes (selbstverständlich nicht validierbares) Konstrukt dienen, das per Accesskey "x" zwei Klick-Events in Folge erzeugt. Der Klick-Event am Button tritt dabei *vor* demjenigen am Link ein:

```
<button onclick="alert('Hui!')">
   <a accesskey="x" href="javascript:alert('Buh!')">
    Hyperlink-Button </a>
</button>
```

Dies »funktioniert« in Opera und Mozilla/Firefox – der Internet Explorer hingegen ignoriert den `<a>`-Tag innerhalb des Buttons regelgerecht und triggert die innere Alert-Box daher nicht.

14.16 caption

Das Element `<caption>` dient im Tabellenkontext dazu, einer Tabelle eine Über- oder Unterschrift hinzuzufügen.

14.16.1 Elementeigenschaften

Einführung	HTML 2.0
Elementtyp	Container
XHTML-Modul	Basic Tables, Tables
Kontext	Das Element tritt optional als erstes unmittelbares Kindelement von `<table>` auf.
Erlaubte Inhalte	Text und Inlineelemente
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	iMode XHTML 1.0, XHTML mobile profile **abgeschafft** in HTML 4.0 mobile, iMode XHTML 1.1

Syntaxbeispiel:

```
<caption> ... </caption>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
caption    { display: table-caption;
             text-align: center; }
```

14.16.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onselectstart`

Elementspezifische Attribute

W3C	`align`
Internet Explorer	`valign`, *Access-*, *Edit-Attribute*

14.16.3 Offizielle Attribute – W3C

align

Wert	*"top"* | *"bottom"* | *"left"* | *"right"*
Typ	festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut legt die Ausrichtung oder Position der Überschrift zur Tabelle fest. Defaultausrichtung ist zentriert oberhalb der Tabelle. Mit Hilfe der vier definierten Werte des Attributs kann die Überschrift wahlweise zentriert über oder unter der Tabelle platziert werden (`"top"`, `"bottom"`), respektive (mit `"left"`, `"right"`) oberhalb der Tabelle rechts- bzw. linksbündig zum Tabellenrand.

Anmerkung: Der Internet Explorer kennt zusätzlich den Wert `"center"`, der für die gleichzeitige Verwendung mit dem IE-Attribut `valign` gedacht ist. Dieser Wert wird in anderen Browsern nicht unterstützt.

Achtung: Mozilla und Firefox verhalten sich für `"left"` und `"right"` unkorrekt und stellen die Caption links bzw. rechts von der Tabelle dar. (Dies entspricht jedoch dem CSS-Verhalten `caption-side:left` bzw. `caption-side:right`.) **[!]**

CSS-Alternativen:

- `caption {caption-side: left|right|top|bottom; }`
  weicht für `left`, `right` von HTML-Verhalten ab.
- `caption {text-align: left|right|center; }`
  entspricht HTML-Verhalten.

14.16.4 Proprietäre Attribute – Internet Explorer

valign

Wert	*"top"* | *"bottom"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut legt im Internet Explorer die vertikale Position der Überschrift zur Tabelle fest. Defaultwert ist oberhalb der Tabelle. Mit Hilfe des Attributs kann die Überschrift wahlweise über oder unter der Tabelle platziert werden. Gleichzeitig kann sie mit dem `align`-Attribut links- oder rechtsbündig ausgerichtet werden.

CSS-Alternative: `caption {caption-side: top | bottom;}`

[!] *Achtung:* Dieses CSS-Property wird im Internet Explorer bis 6.0 nicht unterstützt!

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.16.5 Erläuterung zum Element

Das `<caption>`-Element fügt einer Datentabelle optional eine Über- oder Unterschrift hinzu. Es wird dafür unmittelbar in den `<table>`-Container eingefügt – nach der Spezifikation darf dies nur *unmittelbar nach* dem `<table>`-Starttag vor eventuellen Spaltendefinitionen mit `<col>` oder `<colgroup>` stehen.

[zB] *Beispiel:*

```
<table border="1" width="200">
<caption>Testcaption</caption>
<tr>
<td>Test</td><td>Test</td><td>Test</td><td>Test</td>
</tr>
 ...
</table>
```

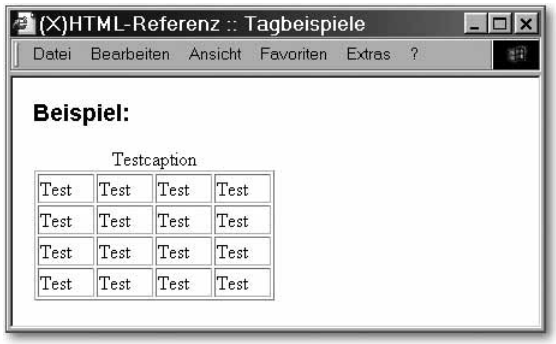

Abbildung 14.18 Tabelle mit Caption in Defaultposition

```
<table border="1" width="200">
<caption align="bottom">Testcaption</caption>
<tr>
<td>Test</td><td>Test</td><td>Test</td><td>Test</td>
</tr>
 ...
</table>
```

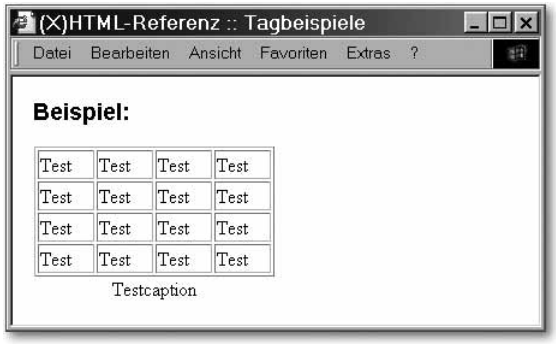

Abbildung 14.19 Tabelle mit Caption in Position »bottom«

Accessibility: Gerade mit Rücksicht auf Screenreader sollte man das `<caption>`-Element nur unmittelbar nach dem Starttag der Tabelle einfügen.

14.17 center

Deprecated: Das `<center>`-Element zentriert die in ihm enthaltenen Inhalte, dient also reinen Präsentationszwecken.

14.17.1 Elementeigenschaften

Einführung	HTML 2.0
Elementtyp	Blockcontainer
XHTML-Modul	Legacy
Kontext	Das Element ist überall innerhalb von Blockelementen und als direktes Kindelement von `<body>` erlaubt.
Erlaubte Inhalte	Alle Block- und Inlineelemente sowie Text.
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0 (tr/fs), 4.01 (tr/fs) XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0, 4.01 strict XHTML 1.0 strict, 1.1, 2.0
Mobil-Spezifikationen	cHTML **abgeschafft** in HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<center> ... </center>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
center   { display: block;
           text-align: center; }
```

14.17.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.17.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.17.4 Erläuterung zum Element

Historisches Element: Das `<center>`-Element zentriert global alle in ihm enthaltenen Elemente (Block- und Inlineelemente) sowie deren Inhalte (Texte, Bilder), sofern nicht für diese anderslautende Ausrichtungsattribute (`align="left"`, `align="right"` oder entsprechende CSS-Properties) vorrangig gelten. Mit dem Element ist keine logische Aussage über dessen Inhalte verbunden, es versieht auch keine strukturierende oder gruppierende Aufgabe.

Als Blockelement steht `<center>` gleichrangig zu `<div>`, ohne aber dessen Strukturbedeutung (»Abschnitt«, »Bereich«) zu besitzen.

Tipp

Verwenden Sie das `<center>`-Element nicht mehr. Funktionstechnisch kann `<div align="center">` als Synonym betrachtet werden und sollte stattdessen eingesetzt werden.

Beispiel: **[zB]**

```
<center>Dieser text ist zentriert.</center>
```

```
<center>
   <p>Ein zentrierter Absatz.</p>
   <p align="left">Linksausrichtung hat hier Vorrang.</p>
```

```
   <p>Wieder ein zentrierter Absatz.</p>
</center>
```

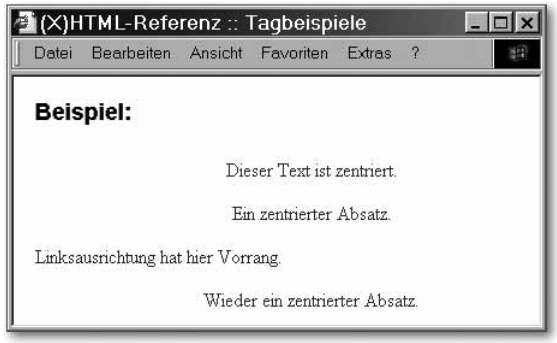

Abbildung 14.20 kann Inline- und Blockelemente enthalten.

14.18 cite

Logische Auszeichnung: Das Element `<cite>` markiert eine Quellenangabe eines Zitats.

14.18.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Das `<cite>`-Element kann beliebig innerhalb von Block- und Inlineelementen eingesetzt werden.
Erlaubte Inhalte	Text und weitere Inlineelemente wie beispielsweise Hyperlinks
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile **abgeschafft** in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<cite> ... </cite>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
cite { font-style: italic }
```

14.18.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.18.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.18.4 Erläuterung zum Element

Das Element `<cite>` markiert eine Quellenangabe, worunter nicht nur die (bibliographische) Herkunft des Zitats, sondern auch eine Urheberangabe (Person oder

Institution) zu verstehen sein kann. Bevorzugt eingesetzt wird das Element innerhalb eines längeren, durch <blockquote> markierten Zitats, es kann jedoch auch innerhalb normaler Textabschnitte als Kennzeichnung einer Quellenangabe verwendet werden:

```
<p><cite>Huckleberry Finn</cite> ist eines der bedeutendsten Werke
der amerikanischen Literatur.</p>
```

[»] *Hinweis:* Das <cite>-Element dient (anders als gelegentlich behauptet) nicht der Markierung von Inlinezitaten – hierfür ist das <q>-Element vorgesehen.

[zB] *Beispiel:*

```
<p>&quot;Nennt mich meinethalben Ismael.&quot; ist ein Zitat aus
<cite>Moby Dick</cite>.</p>

<blockquote cite="http://www.loremipsum.de">
   Suspendisse quis ante. Proin sem nisl, vestibulum a,
   ultrices nec, vehicula nec, sapien.
<cite>Aus: <a href="http://www.loremipsum.de">
http://www.loremipsum.de</a></cite>
</blockquote>
```

Abbildung 14.21 <cite> innerhalb eines Textabsatzes und eines <blockquote>-Containers

14.19 code

Logische Auszeichnung: Das Element <code> markiert den eingeschlossenen Text als Quellcode.

14.19.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Das Element darf in allen Block- und Inlineelementen auftreten.
Erlaubte Inhalte	Das Element darf Inlineelemente und Text enthalten.
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile **abgeschafft** in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<code> ... </code>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
code    { display: inline;
          font-family: monospace; }
```

14.19.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.19.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute:

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute:

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.19.4 Erläuterung zum Element

Das Element `<code>` gehört zu den HTML-Elementen der ersten Generation. Es dient der logischen Auszeichnung seines Inhalts als Quelltext-Abschnitt einer (beliebigen) Programmiersprache. Eine Endmarke `</code>` ist obligatorisch.

Im Gegensatz zu `<pre>` verhält sich `<code>` in Bezug auf Weißraum wie alle anderen HTML-Elemente – im Quelltext erfolgte Einrückungen und Zeilenumbrüche bleiben daher in der Browserdarstellung nicht erhalten, sondern müssen künstlich erzeugt werden. Das Element ist daher besser geeignet für kurze Quellcodeabschnitte (wie die Auszeichnung von Programmbefehlen innerhalb eines Fließtextes) als für die Darstellung längerer Codeabschnitte (verwenden Sie dafür `<pre>`).

```
<p>Dies ist ein JavaScript-Befehl:
   <code>alert('Hallo Welt!')</code>, der einen modalen Dialog
öffnet.</p>
```

Die Präsentation der Inhalte von `<code>` durch den Browser geschieht traditionell in einer Serifenschrift konstanter Laufweite, wie sie zur Quelltextdarstellung geeignet ist (Courier oder Courier New). Sehr kurze Beispiele können Sie auch mit `<samp>` als »sample« auszeichnen.

Beispiel: **[zB]**

```
<p>Ein Funktionsaufruf erfolgt in Javascript über den
Funktionsnamen: <code>meine_funktion()</code>. Die Funktion kann
dabei eine Reihe Parameter übergeben bekommen:</p>

<p><code>meine_funktion(param1, param2)</code></p>

<p>Sie sehen, es ist ganz einfach!</p>
```

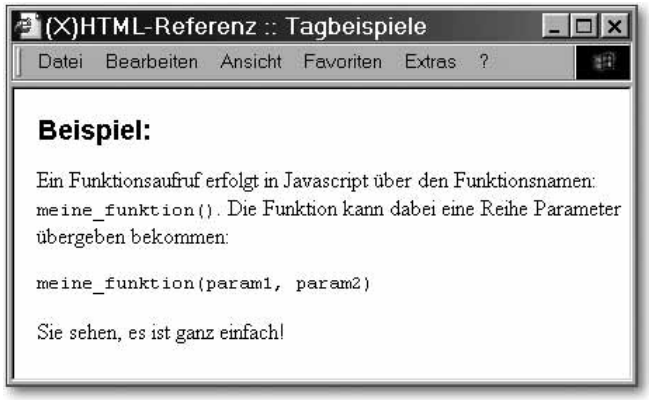

Abbildung 14.22 <code> dient als logische Auszeichnung von Quellcodefragmenten

14.20 col

Das <col>-Element definiert zu Beginn einer Tabelle innerhalb eines <colgroup>-Containers die Eigenschaften einer oder mehrerer Spalten, um dem Browser das nachfolgende Rendering zu erleichtern.

14.20.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Container
XHTML-Modul	Tables
Kontext	Innerhalb von <colgroup> im <table>-Container
Erlaubte Inhalte	Leeres Element
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– **nicht enthalten** in HTML 4.0 mobile, XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

`<col>` (HTML)

`<col />` (XHTML)

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
col          { display: table-column; }
```

14.20.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onreadystatechange`

Elementspezifische Attribute

W3C	`align`, `char`, `charoff`, `span`, `valign`, `width`
Internet Explorer	`bgcolor`

14.20.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt spaltenweit die horizontale Ausrichtung der Inhalte der Tabellenzellen, die der Spalte des `<col>`-Elements zugeordnet sind. Defaultwert ist `"left"`. Ist der Wirkungsbereich des `<cols>`-Elements durch das `span`-

Attribut auf mehrere Spalten erweitert, so bezieht sich der `align`-Wert auf alle diese Spalten.

- `align="left"` (Default)
  richtet die Inhalte aller Zellen der Spalte linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert die Inhalte aller Zellen der Spalte innerhalb des Zellraumes.
- `align="right"`
  richtet die Inhalte aller Zellen der Spalte rechtsbündig zum Zellenrand aus.
- `align="justify"`
  richtet die Inhalte aller Zellen blocksatzmäßig aus.
- `align="char"`
  richtet die Inhalte aller Zellen anhand eines Zeichens relativ zueinander aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann mittels des `char`-Attributs bestimmt werden.
  Achtung: Dieser Attributwert wird derzeit noch nirgends unterstützt. **[!]**

Anmerkung: Das `align`-Attribut für das `<col>`-Element wird derzeit von Opera und Internet Explorer unterstützt, *nicht jedoch* von Mozilla/Firefox.

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem die Inhalte der Zellen der Tabellenspalte durch `align="char"` gegeneinander ausgerichtet werden sollen. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Defaultwert ist je nach Spracheinstellung mittels `lang`-Attribut oder ähnlich der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Anmerkung: Das `char`-Attribut und die Option `align="char"` wird derzeit noch nicht unterstützt.

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das charoff-Attribut bestimmt einen Offset (in Pixel oder Prozent) vom linken Zellrand für das über das char-Attribut des Elements bezeichnete Ausrichtungszeichen. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der charoff-Position ausgerichtet.

Anmerkung: In der Praxis wird das charoff-Attribut ebenso wie char und align="char" noch nicht unterstützt.

span

Wert	*number*
Typ	Numerisch; eine Ganzzahl größer gleich 1
Verwendung	Optional

Das span-Attribut erweitert den Wirkungsbereich der Definitionen des <col>-Elements auf benachbarte Spalten, die in diesem Fall identische Eigenschaften zugewiesen bekommen.

<col **span="3"** align="right"> definiert daher drei gleichartige Spalten. Die Wirkung ist identisch mit der dreimaligen Wiederholung eines <col>-Elements.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das valign-Attribut bestimmt die vertikale Ausrichtung der Inhalte aller Zellen der definierten Tabellenspalte. Defaultwert ist "middle", also die vertikale Zentrierung der Inhalte innerhalb der Zelle.

- valign="top"
  richtet die Inhalte aller Zellen der Spalte am oberen Zellenrand aus.
- valign="middle" (Default)
  zentriert die Inhalte aller Zellen der Spalte vertikal im Zellraum.
- valign="bottom"
  richtet die Inhalte aller Zellen der Spalte am unteren Zellenrand aus.
- valign="baseline"
  richtet die Inhalte aller Zellen nach einer gemeinsamen Textbasislinie aus. Dieser Attributwert wirkt zeilenorientiert auf alle Zellen, in deren Zeilen eine

Zelle mit diesem Wert auftritt. Im Falle einer Spaltendefinition wäre damit automatisch die gesamte Tabelle betroffen.

Anmerkung: Das `valign`-Attribut für das `<col>`-Element wird derzeit von Opera und Internet Explorer unterstützt, *nicht jedoch* von Mozilla/Firefox.

width

Wert	*number[%][*]*
Typ	Breitenangabe in Pixel, Prozent oder Breitenverhältnis
Verwendung	Optional

Deprecated: Das `width`-Attribut bestimmt die Breite der Tabellenspalte in Pixel – z. B. `<col width="100" />` – oder als Prozent der gesamten Tabellenbreite – z. B. `<col width="30%" />`.

In Verbindung mit dem Asterisk-Zeichen kann auch ein Breitenverhältnis mehrerer Spalten angegeben werden. Dieses Beispiel erzeugt drei Spalten im Breitenverhältnis 2:3:4 (die absolute Breite richtet sich nach der Tabellenbreite):

```
<col width="2*" /><col width="3*" /><col width="4*" />
```

Anmerkung: Das `width`-Attribut wird von aktuellen Browsern unterstützt. An seiner Stelle sollten jedoch CSS-Angaben bevorzugt werden.

14.20.4 Proprietäre Attribute – Internet Explorer

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bgcolor` bestimmt im Internet Explorer die Hintergrundfarbe der Tabellenspalte, auf die sich das `<col>`-Element bezieht.

```
<col bgcolor="#ffff00" />
```

14.20.5 Erläuterung zum Element

Mit Hilfe des leeren `<col>`-Elements können den Tabellenzellen einer Spalte gemeinsame Eigenschaften zugewiesen werden, was Zellenbreite sowie die horizontale und vertikale Ausrichtung der Zellinhalte betrifft. In HTML ist eine **Endmarke** für das Element verboten. In XHTML muss es dagegen stets als `<col/>` mit innerer Abschlussmarke geschrieben werden.

Eine Folge von <col>-Elementen in beliebiger Zahl darf als unmittelbare Kindelemente des <table>-Containers erscheinen, wobei diesen lediglich ein optionales <caption>-Element vorangehen darf.

```
<table border="0" width="640">
   <caption>Tabellen&uuml;berschrift</caption>
   <col width="140" span="2" />
   <col width="360" />
   <col width="140" />
   <tr>
      <td> ...</td>
      <td> ... </td>
      <td> ... </td>
   </tr>
...
</table>
```

Alternativ können <col>-Elemente innerhalb von <colgroup>-Containern erscheinen.

```
<table border="0" width="640">
   <colgroup>
      <col width="200" />
      <col width="150" />
      <col width="150" />
   </colgroup>
   <tr>
      <td> ...</td>
      <td> ... </td>
      <td> ... </td>
   </tr>
...
</table>
```

Eine Folge aus Spaltendefinitionen mit <col>-Elementen, die nicht innerhalb eines <colgroup>-Containers stehen, bilden jeweils eine implizite Spaltengruppe, als ob sie von einem <colgroup>-Container umgeben wären. <col>- und <colgroup>-Elemente dürfen daher auf gleicher Hierarchieebene auch gemischt werden.

Im folgenden Beispiel sind drei Spaltengruppen definiert, wobei die erste eine Spalte, die zweite drei Spalten und die dritte zwei Spalten beinhaltet:

```
<table border="0" width="640">
   <!-- erste, implizite Gruppe: -->
   <col width="150" />
```

```
   <!-- zweite, explizite Gruppe: -->
   <colgroup align="right">
      <col width="50" />
      <col width="50" />
      <col width="50" />
   </colgroup>
   <!-- dritte, implizite Gruppe: -->
   <col width="150" />
   <col width="150" />
   <tr>
      <td> ...</td>
      <td> ... </td>
      <td> ... </td>
   </tr>
...
</table>
```

14.21 colgroup

Das `<colgroup>`-Element bestimmt die Eigenschaften mehrerer Tabellenspalten gleichzeitig als so genannte Spaltengruppe (*column group*) oder enthält `<col>`-Elemente zur Definition der Eigenschaften einzelner Spalten.

14.21.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Container
XHTML-Modul	Tables
Kontext	Innerhalb des `<table>`-Containers vor der ersten Tabellenzeile `<tr>`
Erlaubte Inhalte	Optional `<col>`-Elemente oder leer
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<colgroup> ... </colgroup>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
colgroup   { display: table-column-group; }
```

14.21.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onreadystatechange`

Elementspezifische Attribute

W3C	`align`, `char`, `charoff`, `span`, `valign`, `width`
Internet Explorer	`bgcolor`

14.21.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt spaltenweit die horizontale Ausrichtung der Inhalte der Tabellenzellen, die der Spalten des `<colgroup>`-Elements zugeordnet sind. Defaultwert ist `"left"`. Ist der Wirkungsbereich des `<colgroup>`-Elements durch das `span`-Attribut auf mehrere Spalten erweitert, so bezieht sich der `align`-Wert auf alle diese Spalten.

- `align="left"` (Default)
  richtet die Inhalte aller Zellen der Spalte linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert die Inhalte aller Zellen der Spalte innerhalb des Zellraumes.
- `align="right"`
  richtet die Inhalte aller Zellen der Spalte rechtsbündig zum Zellenrand aus.

- `align="justify"`
  richtet die Inhalte aller Zellen blocksatzmäßig aus.
- `align="char"`
  richtet die Inhalte aller Zellen anhand eines Zeichens relativ zueinander aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann mittels des `char`-Attributs bestimmt werden.

  Achtung: Dieser Attributwert wird derzeit noch nirgends unterstützt. **[!]**

Anmerkung: Das `align`-Attribut für das `<colgroup>`-Element wird derzeit von Opera und Internet Explorer unterstützt, *nicht jedoch* von Mozilla/Firefox.

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem die Inhalte der Zellen der Tabellenspalte durch `align="char"` gegeneinander ausgerichtet werden sollen. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Defaultwert ist je nach Spracheinstellung mittels `lang`-Attribut oder ähnlich der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Anmerkung: Das `char`-Attribut und die Option `align="char"` wird derzeit noch nicht unterstützt.

charoff

Wert	*number[%]*
Typ	Ganzzahl oder Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent vom linken Zellrand für das Ausrichtungszeichen. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

Anmerkung: In der Praxis wird das `charoff`-Attribut, ebenso wie `char` und `align="char"` noch nicht unterstützt.

span

Wert	*number*
Typ	Numerisch; eine Ganzzahl größer gleich 1
Verwendung	Optional

Das `span`-Attribut ermöglicht die Definition mehrerer Spalten durch ein `<colgroup>`-Element, ohne dass dieses `<col>`-Elemente beinhalten muss. Alle so erzeugten Spalten erhalten die gleichen Eigenschaften bezogen auf benachbarte Spalten, die in diesem Fall identische Eigenschaften zugewiesen bekommen.

`<colgroup span="3" align="right" />` definiert drei gleichartige Spalten. Die Wirkung ist identisch mit der dreimaligen Wiederholung eines `<col>`-Elements im Inneren des `<colgroup>`-Containers.

Anmerkung: Sind `<col>`-Elemente im `<colgroup>`-Container enthalten und weicht ihre Anzahl von der mittels des `span`-Attributs bezeichneten ab, so hat die Anzahl der `<col>`-Elemente Vorrang.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung der Inhalte aller Zellen der definierten Tabellenspalten. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zelle.

- `valign="top"`
  richtet die Inhalte aller Zellen der Spalten am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert die Inhalte aller Zellen der Spalten vertikal im Zellraum.
- `valign="bottom"`
  richtet die Inhalte aller Zellen der Spalten am unteren Zellenrand aus.
- `valign="baseline"`
  richtet die Inhalte aller Zellen nach einer gemeinsamen Textbasislinie aus. Dieser Attributwert wirkt zeilenorientiert auf alle Zellen, in deren Zeilen eine Zelle mit diesem Wert auftritt. Im Falle einer Spaltendefinition wäre damit automatisch die gesamte Tabelle betroffen.

Anmerkung: Das `valign`-Attribut für das `<colgroup>`-Element wird derzeit von Opera und Internet Explorer unterstützt, *nicht jedoch* von Mozilla/Firefox.

width

Wert	*number*
Typ	Breitenangabe in Pixel oder Prozent
Verwendung	Optional

Deprecated: Das `width`-Attribut bestimmt die Breite aller definierten Tabellenspalten in Pixel, als Prozent der gesamten Tabellenbreite oder in Form eines konstanten Breitenverhältnisses.

```
<colgroup width="300"><col/><col/><col/></colgroup>
```

setzt beispielsweise alle drei Spaltendefinitionen auf eine Breite von 300 Pixel. Mittels der innenliegenden `<col>`-Elemente kann jede Spalte zusätzlich weitere Eigenschaften erhalten. (Die Vergabe einer Breite am `<col>`-Element hätte dabei Vorrang vor derjenigen, die durch das `<colgroup>`-Element bestimmt ist.)

```
<colgroup width="*" span="3" />
```

erzeugt drei Spalten gleicher Breite. (Angaben wie `"2*"`, `"3*"` etc. machen an dieser Stelle keinen Sinn, da auch dies ein konstantes Breitenverhältnis erzeugt.)

Anmerkung: Das `width`-Attribut wird von allen aktuellen Browsern unterstützt. An seiner Stelle sollten jedoch CSS-Angaben bevorzugt werden.

14.21.4 Proprietäre Attribute – Internet Explorer

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bgcolor` bestimmt im Internet Explorer die Hintergrundfarbe der Tabellenspalten, auf die sich das `<colgroup>`-Element bezieht.

```
<colgroup bgcolor="#dddddd"> ... </colgroup>
```

14.21.5 Erläuterung zum Element

Das `<colgroup>`-Element gruppiert in einer Tabelle `<col>`-Elemente in beliebiger Anzahl zu jeweils einer Spaltengruppe. Die **Endmarke** `</colgroup>` ist in HTML optional, doch in XHTML vorgeschrieben. Das Element ist unmittelbares Kindelement von `<table>`, wobei ihm das optionale `<caption>`-Element oder beliebig

viele <col>-Elemente einer (dann impliziten) Spaltengruppe vorangehen können.

Explizite Spaltengruppen mit <colgroup>:

Hier ein Beispiel für eine Tabelle mit zwei durch <colgroup> gebildeten Spaltengruppen aus je zwei Spalten:

```
<table border="0" width="640">
   <colgroup>
      <col width="140">
      <col width="360">
   </colgroup>
   <colgroup>
      <col width="140">
      <col width="360">
   </colgroup>
   <tbody>
...
  </tbody>
</table>
```

Da die Zahl der in <colgroup> einzufügenden <col>-Elemente beliebig ist, kann das Element auch leer bleiben, wobei die Zahl der zu definierenden Spalten durch das span-Attribut festgelegt wird.

Folgende drei Definitionen sind identisch und beschreiben je drei rechts ausgerichtete Spalten:

```
<colgroup span="3" align="right" />
<colgroup span="3" align="right"></colgroup>
<colgroup align="right">
   <col/><col/><col/>
</colgroup>
```

Sobald ein <colgroup>-Element mindestens ein <col>-Element enthält, ist der User Agent dazu angehalten, ein eventuelles span-Attribut in <colgroup> zu *ignorieren*.

Die folgende Definition ergibt daher ebenfalls drei Tabellenspalten:

```
<colgroup span="5" align="right">
   <col/><col/><col/>
</colgroup>
```

Implizite Spaltengruppen

Ähnlich wie Tabellenzeilen stets einem (gegebenenfalls impliziten) Tabellenrumpf `<tbody>` zugerechnet werden, bilden benachbarte `<col>`-Geschwisterelemente, die nicht innerhalb eines `<colgroup>`-Containers stehen, für sich dennoch eine Spaltengruppe.

Auch im folgenden Beispiel sind daher zwei Spaltengruppen definiert, wobei die erste explizit, die zweite implizit ist:

```
<table border="0" width="640">
   <colgroup>
      <col width="140">
      <col width="360">
   </colgroup>
   <col width="140">
   <col width="360">
   <tbody>
...
  </tbody>
</table>
```

14.22 dd

Das Element `<dd>` kennzeichnet die Definition (*definition data*) zu einem Begriff (*definition term*) innerhalb einer Definitionsliste.

14.22.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	List
Kontext	Innerhalb einer Definitionsliste `<dl>`
Erlaubte Inhalte	Block-, Inlineelemente und Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<dd> ... </dd>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
dd    { display: block;
        margin-left: 40px;
   /*   entspricht em-Marginbreite:  */
   /*   margin-left: 4em;            */
         }
```

14.22.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	`nowrap`, *Access-, Edit-Attribute*

14.22.3 Proprietäre Attribute – Internet Explorer

nowrap

Wert	*"nowrap"* | *"true"* | *"false"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	IE4

Das nowrap-Attribut erlaubt bzw. unterdrückt den automatischen Zeilenumbruch innerhalb des Elements. Defaultmäßig bricht der Browser lange Zeilen automatisch um. Der Wert nowrap="false" gestattet ihm dies (überflüssigerweise) explizit, während nowrap="true" bzw. nowrap="nowrap" den Umbruch verbieten. Gegebenenfalls erscheint hierdurch eine horizontale Scrollleiste.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

accesskey = character (IE 5)

hidefocus = "true" | "false" (IE 5.5)

tabindex = number (IE 5)

IE-Edit-Attribute

atomicselection = "true" | "false" (IE 5)

contenteditable = "inherit" (Default) | "false" | "true"

unselectable = "on" | "off" (IE 5)

14.22.4 Erläuterung zum Element

Der Container <dd> (*definition data*) enthält die Definition eines Begriffs innerhalb einer Definitionsliste <dl>. Die Definition bezieht sich dabei auf den in der Liste unmittelbar vorangehenden Definitionsterm <dt>. Die Endmarke </dd> ist in HTML optional und in XHTML vorgeschrieben.

Der erlaubte Elementinhalt umfasst sowohl Block- als auch Inlineelemente, darunter auch Definitionslisten, so dass ein <dd>-Container selbst ebenfalls weitere <dl>-Container enthalten darf. Dies entspricht der Verschachtelungsmöglichkeit von geordneten und ungeordneten Listen.

Beispiel: **[zB]**

Die folgende Definitionsliste enthält drei Begriffe und deren Definition:

```
<dl>
   <dt>Gef&auml;hrlich:</dt>
      <dd>Einen Boxer reizen</dd>
   <dt>Sehr gef&auml;hrlich:</dt>
      <dd>Zwei Boxer reizen</dd>
   <dt>Höchst gef&auml;hrlich:</dt>
      <dd>Einen Pudel reizen</dd>
</dl>
```

Abbildung 14.23 Eine Definitionsliste <dl> mit Definitionen <dt> und <dd>

Querverweis: Weitere Informationen zu Definitionslisten finden Sie bei der ausführlichen Beschreibung des `<dl>`-Containers.

14.23 del

Logische Auszeichnung: Markiert eingeschlossene Inhalte im Sinne einer Änderungsmarkierung als gelöscht bzw. entfernt.

14.23.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Edit
Kontext	Überall in Blockelementen
Erlaubte Inhalte	Text und Inlineelemente
Offizielle W3C Spezifikationen	HTML 4.0, 4.01, XHTML 1.0, 1.1; **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	– **nicht enthalten** in HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<del> ... </del>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
del   { display: inline;
        text-decoration: line-through;    /* vorgesehen */
        }
```

14.23.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`cite`, `datetime`
Internet Explorer	*Access-, Edit-Attribute*

14.23.3 Offizielle Attribute – W3C

cite

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `cite`-Attribut enthält einen URI als Quelle für Angabe von Gründen für die Änderungen bzw. über den Urheber.

Achtung: Das `cite`-Attribut wird im Internet Explorer nicht unterstützt. **[!]**

datetime

Wert	*cdata*
Typ	Ein String in Form eines Zeitstempelwertes oder Datums
Verwendung	Optional

Das datetime-Attribut enthält Angaben über den Zeitpunkt der Änderung in Form eines Datums- oder Zeitstempelwertes. Das Attribut ist *case-sensitive*.

```
<!-- gelöscht am 26. Oktober 2006: -->
<del datetime="2006-10-26"> ... </del>
```

14.23.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.23.5 Erläuterung zum Element

Das Element dient der Markierung von Textpassagen, die im Rahmen einer Überarbeitung gelöscht wurden. Gelöschte Textstellen werden üblicherweise durchgestrichen dargestellt. Mittels des datetime-Attributs kann der Zeitpunkt der Änderung angegeben werden, wobei ein Datums- oder ein Zeitstempelwert übergeben werden kann. Das cite-Attribut nennt optional den Urheber der Änderung oder verweist auf den Änderungsgrund (beispielsweise mittels eines URI, der dann jedoch nur dokumentarische und nicht funktionale Bedeutung besitzt).

Das Element fungiert gewöhnlich als **Inlinecontainer** innerhalb eines Blockcontainers. Ebenso ist es jedoch möglich, Quelltextpassagen aus mehreren Blockelementen als Einfügung auszuzeichnen, weshalb <del> ausdrücklich auch auf Blocklevel-Ebene erscheinen darf (als unmittelbarer Inhalt des <body>-Elements) und damit quasi die Funktion eines Blockelements übernimmt.

[zB] *Beispiel:*

```
<p>Dies ist Text, wobei einige Passagen
   <del>gel&ouml;scht</del>
   <ins>eingef&uuml;gt</ins>
wurden.</p>
```

Abbildung 14.24 Präsentation von <ins> und <del> im Internet Explorer

Anmerkung: Das Element `<del>` als explizite Löschmarke *entfällt* in XHTML 2.0. Seine Aufgabe kann (besser) von dem neuen Standardattribut `edit` übernommen werden, dessen erlaubte Werte `"inserted"`, `"deleted"`, `"changed"` und `"moved"` es ermöglichen, jedem Elementcontainer Informationen über inhaltliche Veränderungen mitzugeben.

Das Attribut `datetime`, das den Zeitpunkt der Änderung in Form eines Zeitstempels vermerkt, kann nun ebenfalls jedem Element zugewiesen werden. Beide Attribute sind Teil der Edit-Collection von XHTML 2.0 und gehören als Teil der Common Collection zu den Standardattributen.

14.24 dfn

Logische Auszeichnung: Das Element `<dfn>` (*definition*) markiert den eingeschlossenen Inhalt als zu definierenden Begriff.

14.24.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile **abgeschafft** in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<dfn> ... </dfn>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
dfn    {font-style: italic; }
```

14.24.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.24.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.24.4 Erläuterung zum Element

Das eher selten genutzte Element `<dfn>` dient zur logischen Kennzeichnung einer Begriffsdefinition oder eines innerhalb eines Textes neu eingeführten Fachwortes. Die Endmarke `</dfn>` ist vorgeschrieben. Die bevorzugte Darstellung des Elementinhalts im Browser ist kursiv, was der gängigen typographischen Gepflogenheit bei der Einführung eines neuen Begriffes folgt.

Das Element wird eingesetzt, um bei der ersten Verwendung eines definitionsbedürftigen Begriffs innerhalb eines Fließtextes diesen hervorzuheben und seine Bedeutung zu erklären. Zu diesem Zweck kann zum einen auf das `title`-Attribut des Elements zurückgegriffen werden. Eine andere, ebenfalls gängige Vorgehensweise ist, die eigentliche Definition im Fließtext folgen zu lassen.

Hinweis: Mit `<dfn>` verwandte logische Auszeichnungen sind `<abbr>`, das eine Abkürzung bezeichnet, und `<acronym>`, das ein Akronym markiert. Beide bedienen sich allerdings ausschließlich des `title`-Attributs zur Begriffserklärung. **[«]**

Beispiel: **[zB]**

```
<p>Dies ist eine <dfn title="Erkl&auml;rung eines
Begriffs">Definition</dfn>, die im Browser kursiv dargestellt wird.
Unter einer Definition versteht man die Erkl&auml;rung eines
Begriffs.</p>
```

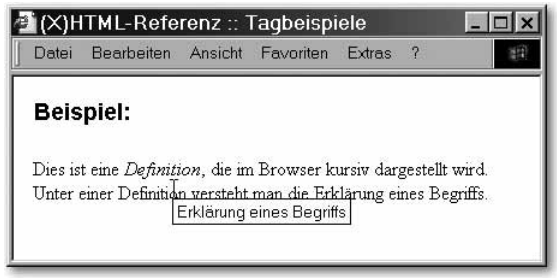

Abbildung 14.25 Darstellung von mit <dfn> markierten Definitionen

14.25 dir

Deprecated: Das (logische) Element `<dir>` (*directory list*) markiert eine Directory- bzw. Verzeichnisliste.

14.25.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Legacy
Kontext	Erlaubt innerhalb von `<applet>`, `<blockquote>`, `<body>`, `<button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<map>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`
Erlaubte Inhalte	Listenelemente `<li>`
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0/4.01 (tr/fs), XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0/4.01 strict, XHTML 1.0 strict, XHTML 1.1, 2.0
Mobil-Spezifikationen	cHTML *nicht enthalten in:* HTML 4.0 mobile, XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<dir> ... </dir>
```

Defaultrendering:
Das Element wird standardmäßig als Block dargestellt.

CSS-Umsetzung des HTML-Defaults:

```
dir    { display:block; margin-left: 40px }
```

14.25.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	compact
Internet Explorer	type, *Access-, Edit-Attribute*
Netscape	type

14.25.3 Offizielle Attribute – W3C

compact

Wert	*"compact"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Deprecated (Präsentationsattribut): Das compact-Attribut reduziert den Abstand zwischen den Listenelementen der Liste und bewirkt eine kompaktere Darstellung der Liste.

Anmerkung: Das compact-Attribut wird in der Praxis von kaum einem Browser unterstützt. Es ist anzuraten, für die beabsichtigte Wirkung auf das CSS-margin-Property der Listenelemente zurückzugreifen.

14.25.4 Proprietäre Attribute – Internet Explorer

type

Wert	*"disc"* | *"circle"* | *"square"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	IE 3

Legt im Internet Explorer die Art des Bullets der vom `<dir>`-Container eingeschlossenen Listenelemente `<li>` fest.

- `type="disc"` (Default)
  erzeugt ein Bullet in Form einer schwarzen Scheibe.
- `type="circle"`
  erzeugt ein Bullet in Form eines Kreises.
- `type="square"`
  erzeugt ein Bullet in Form eines schwarzen Quadrats.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.25.5 Proprietäre Attribute – Netscape

type

Wert	*"disc"* | *"circle"* | *"square"*
Typ	Festgelegter Bezeichner
Verwendung	Optional
Einführung	NN4

(Historisch) Legt im Netscape Navigator 4.0 die Art des Bullets der vom Element eingeschlossenen Listenelemente `<li>` fest.

- `type="disc"` (Default)
  erzeugt ein Bullet in Form einer schwarzen Scheibe.
- `type="circle"`
  erzeugt ein Bullet in Form eines Kreises.
- `type="square"`
  erzeugt ein Bullet in Form eines schwarzen Quadrats.

14.25.6 Erläuterung zum Element

Das <dir>-Element war ursprünglich zum Erzeugen **mehrspaltiger Verzeichnislisten** gedacht, wobei eine Reihe kurzer Einträge in Spaltenform nebeneinander arrangiert werden sollte. Die Spezifikation von HTML 2.0 ging von einer Maximallänge von zwanzig Zeichen pro Eintrag und einer Spaltenbreite von 24 Zeichen aus, was jedoch durch den User Agent optimiert werden sollte.

Beispiel: (entlehnt aus der Spezifikation zu HTML 2.0) **[zB]**

```
<DIR><LI>A-H<LI>I-M<LI>M-R<LI>S-Z</DIR>
```

In der Praxis wurde eine <dir>-Liste von allen Browsern exakt wie eine -Liste dargestellt, weshalb das Element nie wie geplant eingesetzt werden konnte. Mittlerweile ist es aus den *strict*-Vokabularen entfernt und kann als abgeschafft gelten.

Beispiel: **[zB]**

```
<dir>
<li>/buch/kapitel_1/</li>
<li>/buch/kapitel_1/abbildungen/</li>
<li>/buch/kapitel_1/beispiele/html/</li>
<li>/buch/kapitel_1/beispiele/images/</li>
</dir>
```

Abbildung 14.26 Eine Directory-Liste mit <dir>

14.26 div

Das -Element beschreibt einen Blockcontainer im Sinne eines logischen Dokumentbereichs, ohne ihm typographische Bedeutung (Absatz o.ä.) zu geben.

14.26.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Blockcontainer
XHTML-Modul	Text
Kontext	Hauptsächlich in `<body>`; außerdem gestattet in `<applet>`, `<blockquote>`, `<button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<map>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`
Erlaubte Inhalte	Block- und Inlineelemente sowie Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile prof., iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<div> ... </div>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
div  { display:block; }
```

14.26.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforecopy`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfilterchange`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onscroll`, `onselectstart`

Elementspezifische Attribute

W3C	`align`
Internet Explorer	`nowrap`, *Access-*, *Databinding-*, *Edit-Attribute*
Netscape	`cols`, `gutter`, `width` (N4)

14.26.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte des `<div>`-Containers. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet die Inhalte linksbündig zum Containerrand aus.
- `align="center"`
  zentriert die Inhalte innerhalb des Containerbereiches.
- `align="right"`
  richtet die Inhalte rechtsbündig zum Containerrand aus.
- `align="justify"`
  richtet (Text-)Inhalte im Container blocksatzmäßig an beiden Rändern aus.

14.26.4 Proprietäre Attribute – Internet Explorer

nowrap

Wert	*"nowrap"* | *"true"* | *"false"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	IE3

Das `nowrap`-Attribut erlaubt bzw. unterdrückt den automatischen Zeilenumbruch innerhalb des Elements. Defaultmäßig bricht der Browser lange Zeilen automatisch um. Der Wert `nowrap="false"` gestattet ihm dies (überflüssigerweise) explizit, während `nowrap="true"` bzw. `nowrap="nowrap"` den Umbruch verbieten. Gegebenenfalls erscheint hierdurch eine horizontale Scrollleiste.

Proprietäre Attributsammlungen – Internet Explorer Proprietäre Attribute

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE)

`dataformatas = "text" | "localized-text" | "HTML"` (IE 4)

`datasrc = "datenquellen-id"`

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.26.5 Proprietäre Attribute Netscape

Die drei historischen Netscape-Attribute `cols`, `gutter` und `width` erlaubten (zumindest in der Theorie), in Netscape 4 den Inhaltsbereich des `<div>`-Containers in vertikale Spalten gleicher Breite zu unterteilen. Dies ist analog zum proprietären Netscape-Element `<multicol>` für mehrspaltigen Textinhalt zu betrachten. Alle drei Attribute werden von neueren Netscape-Versionen nicht mehr unterstützt.

cols

Wert	*number*
Typ	Ganzzahl als Festlegung der Spaltenzahl
Verwendung	Optional
Einführung	N4

(Historisch, proprietär) Das `cols`-Attribut bestimmt die Anzahl der vertikalen Inhaltsspalten gleicher Breite, in die der Inhaltsbereich des `<div>`-Containers zu unterteilen ist. Der Browser ist angehalten, die Inhalte so auf die Spalten zu verteilen, dass eine weitestgehend identische Kolumnenhöhe erzielt wird.

gutter

Wert	*number*
Typ	Ganzzahl als Pixelwert; bestimmt Spaltenzwischenraum
Verwendung	Optional
Einführung	N4

(Historisch, proprietär) Das `gutter`-Attribut definiert wie das Cellspacing bei Tabellen den freien Raum zwischen den Inhaltsspalten. Der Defaultwert liegt bei 10 Pixel; er kann mittels `gutter` vergrößert oder verkleinert werden.

width

Wert	*number[%]*
Typ	Ganzzahl als Pixelwert oder Prozentangabe
Verwendung	Optional
Einführung	N4

(Historisch, proprietär) Das `width`-Attribut bestimmt die Breite einer einzelnen Inhaltsspalte (und damit die Breite aller) mittels eines Pixelwertes oder einer Prozentangabe, die sich auf die Gesamtbreite des Containerinneren bezieht. Ist kein `width`-Attribut vergeben, wird stets der ganze zur Verfügung stehende Raum aufgeteilt.

Anmerkung: Das `width`-Attribut bezieht sich *nicht* auf die Gesamtbreite *aller* Spalten. Die diesbezügliche Annahme beruht auf einer Fehlinterpretation des `width`-Attributs von `<multicol>` durch Netscape 4. Auch dort jedoch entsprach die ursprüngliche Intention der oben beschriebenen.

14.26.6 Erläuterung zum Element

Mittels des `<div>`-Containers definiert man logische Bereiche (*divisions*) innerhalb eines HTML-Dokuments. Im zeitgenössischen Webdesign kann `<div>` mit Fug und Recht als das zentrale Element angesehen werden, das anstelle des überkommenen Tabellenlayouts die strukturelle Grundlage für ein Sitedesign bildet.

Styling per CSS

Analog zu `<span>`, das einen neutralen Inline-Container darstellt, bildet `<div>` einen neutralen Blockcontainer, der außer dem üblichen Zeilenumbruch vor und nach dem Element keine weitere Präsentation im Browser bewirkt. Aufgrund der Darstellungsneutralität des Containers werden Präsentationen per CSS an ihn

gebunden. Hierfür ist es sinnvoll, den -Container entsprechend zu markieren.

Einen CSS-Elementselektor einzusetzen, um allen -Containern eine gleichartige Präsentation zuzuweisen, ist nicht sinnvoll, da das Element normalerweise in mehreren Funktionen im Einsatz ist. Verwenden Sie

- id-Identifier und ID-Selektor für nicht-wiederholbare Strukturen.
- class-Attribut und eine Stylesheetklasse für wiederholbare Strukturen.

Nicht wiederholbar sind die logischen Abschnitte wie Navigation und Inhaltsbereich, aus denen ein Dokument zusammengesetzt ist,. Je nach Bedarf können weitere Bereiche erforderlich sein (Kopf- und Fußbereiche, Kontextbereiche).

Als wiederholbar gelten Abschnitte, die in der Zahl ihres Auftretens unbestimmt sind, wie Gruppierungen von Inhalten (die beispielsweise einen Artikelanreißer mit Überschrift, Textbeschreibung und Bild zusammenfassen). Besonders wenn Sie Seiteninhalte aus Datenbanken generieren, sollten Sie auf Stylesheetklassen zurückgreifen.

Mittels CSS kann einer durch -Container gebildeten Struktur eine beliebige Präsentation gegeben werden, indem die Elementcontainer mit Abmessungen (und weiteren Eigenschaften) versehen und gegeneinander positioniert werden. Da als Inhalte von sämtliche Block- und Inlineelemente (inklusive selbst) erlaubt sind, kann das Element eingesetzt werden, um beliebige Gruppierungen anderer Strukturen zu bilden.

Dokumentstruktur mit

Für eine Dokumentstruktur, an die ein Layout gebunden werden soll, wird unmittelbar im <body>-Element eingesetzt und die Struktur aus mehreren aufeinanderfolgenden -Containern aufgebaut. Solche **Strukturcontainer** erhalten einen Identifier mittels des id-Attributs, um sie per CSS erfassen und nach Wunsch präsentieren zu können. Die Quelltextreihenfolge sollte so gewählt werden, dass die Inhalte auch ohne CSS-Präsentation nutzbar bleiben.

```
...
<body>
<!-- Kopfbereich mit Logo, im Fluss -->
<div id="kopfbereich"></div>
<!-- Menübereich mit Navigation; links gefloatet -->
<div id="menuebereich"></div>
<!-- Inhalt; im Fluss rechts v. d. Navigation -->
<div id="inhaltsbereich"></div>
<!-- Fußbereich mit Copyright o.ä.; im Fluss -->
```

```
<div id="fussbereich"></div>
</body>
...
```

Listing 14.4 Einfache, mit -Containern erzeugte Dokumentstruktur

Floaten von Seitenbereichen

Der Navigationsbereich kann einfach mit `float:left` am Seitenrand positioniert werden, wobei Kopf-, Inhalt- und Fußbereich im normalen Fluss verbleiben. Die Bereichsabmessungen können per CSS an den User Agent angepasst werden, mittels alternativer Stylesheets für Screen- oder Mobilanwendungen also unterschiedlich gestaltet werden. Gleiches wäre mit einem Tabellenlayout nicht möglich. Wird das Dokument ohne Stylesheetinformationen betrachtet (wie es bei einem Textbrowser und vielen Screenreadern der Fall ist), kommt die Quelltextreihenfolge zum Tragen.

Bei den CSS-Werten ist hauptsächlich darauf zu achten, dass der Inhaltsbereich einen linken Marginrand erhält, welcher der Breite des Menübereichs entspricht, oder diese leicht übersteigt (in Abhängigkeit von den restlichen Box-Properties der Bereiche).

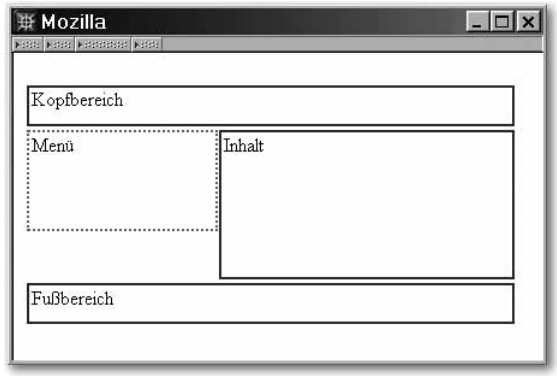

Abbildung 14.27 Links gefloateter Menübereich (gepunktet)

Anmerkung: Hier und in Folge sind Screenshot taugliche Beispielwerte für die Breiten eingesetzt, die nur der Verdeutlichung dienen. Praxiswerte wären um ca. den Faktor 3 größer.

```
#menuebereich   {
   float:left;
   border:2px dotted red;
   width:9.6em;
}
```

```
#inhaltsbereich  {
   border:2px solid blue;
   width:15em;
   margin-left:10em;
}
```

Listing 14.5 Relevante CSS-Properties für Float-Layout (Auszug)

Der *Nachteil* dieses Prinzips, der besonders bei mehrspaltigem Layout (links gefloatete Navigation, Inhaltsspalte, rechts gefloatete Kontextspalte) ist, dass die sekundären Inhalte (Navigation, Kontextspalte), sobald sie gefloatet werden, im Quelltext stets *vor* den relevanten Inhalten erscheinen müssen. Dies erschwert besonders den Suchmaschinen das Erkennen der durch sie eigentlich zu indexierenden Inhalte, da zunächst Navigations- und Kontextbereich eingelesen werden.

Absolute Positionierung von Seitenbereichen

Nicht immer ist daher eine Darstellung der Inhalte in Quelltextreihenfolge sinnvoll. Speziell im Hinblick auf eine Optimierung für Suchmaschinen kann es nutzbringend sein, die Textinhalte möglichst weit vorn im Quelltext zu platzieren und Kontext- und Navigationsbereiche erst im Anschluss an diese.

Durch CSS-Positionierung werden die im Quelltext nach hinten verschobenen Bereiche optisch an die gewünschte Position gerückt. Hierbei ist zunächst darauf zu achten, dass die im Fluss bleibenden Seitenelemente über CSS-Margins den entsprechenden Raum freimachen.

```
...
<body>
<!-- Kopfbereich mit Logo; im Fluss -->
<div id="kopfbereich"></div>
<!-- vorgezogener Inhalt; im Fluss -->
<div id="inhaltsbereich"></div>
<!-- Menübereich mit Navigation; absolut positioniert-->
<div id="menuebereich"></div>
<!-- Fußbereich mit Copyright o.ä.; im Fluss -->
<div id="fussbereich"></div>
</body>
...
```

Listing 14.6 Mit -Containern erzeugte Dokumentstruktur; Inhalt vor Navigation

Die Präsentation mit CSS erfolgt ansonsten weitestgehend analog zum Floatlayout – mit Ausnahme des Navigationsbereichs, für den diesmal eine absolute Position angegeben wird. Der Menübereich muss mit dem `top`-Property lediglich ent-

sprechend der Höhe des Kopfbereiches gegenüber dem oberen Seitenrand positioniert werden:

```
#menuebereich  {
   position:absolute;
   top:2.8em;
   border:2px dashed red;
   height:6em;
   width:9.6em;
}

#inhaltsbereich  {
   border:2px solid blue;
   height:5.5em;
   width:15em;
   margin-left:10em;
}
```

Listing 14.7 Relevante CSS-Properties für Positioning-Layout (Auszug)

Der *Nachteil* dieser Vorgehensweise besteht darin, dass mit `position:absolute` positionierte Bereiche aus den Dokumentfluss genommen sind. Im Gegensatz zu Float-Layouts ist es nicht möglich, die im Textfluss verbliebenen Elemente stets automatisch korrekt gegenüber den positionierten Bereichen zu platzieren. Dies gilt beispielsweise einen Fußbereich, der unterhalb des Inhaltsbereiches erscheinen soll, sobald für diesen keine Mindesthöhe definiert ist bzw. er die Höhe der absolut positionierten Navigation unterschreitet (siehe Abbildung 14.28).

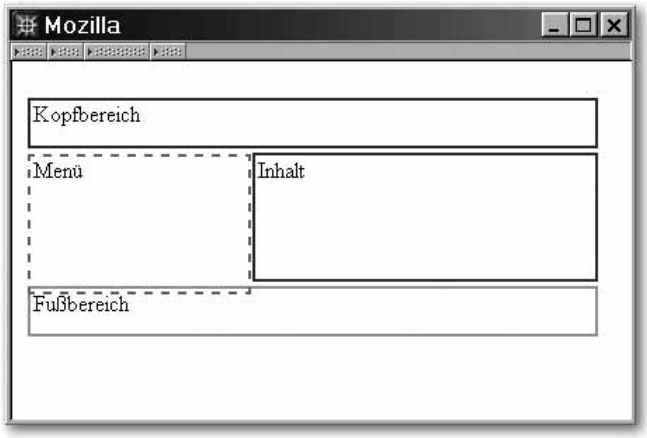

Abbildung 14.28 Absolut positionierter Menübereich (gestrichelt)

Bei der Repositionierung von Inhalten ist es sinnvoll, am Seitenanfang Sprunglinks einzusetzen, die die Navigation innerhalb des Dokuments erleichtern, sobald dieses ohne CSS-Präsentation gezeigt wird (Sprung vom Seitenanfang zur Navigation oder zu den Inhalten)[22]. Als logischer Container um die Sprunglinks wird wiederum ein `<div>` eingesetzt. Linkziel ist der Identifier des jeweiligen Bereichscontainers:

```
...
<body>
<div id="sprunglinks">
   <a href="#menuebereich">zur Navigation</a> |
   <a href="#inhaltsbereich">zum Inhalt</a>
</div>

<div id="kopfbereich"></div>

<div id="inhaltsbereich">
   <!-- viel Inhalt hier: -->
   Hier steht eine Menge Inhalt, an dem ...
</div>

<div id="menuebereich">
   <ul>
      <li><a href="eins.html">erste Seite</a></li>
      <li><a href="zwei.html">zweite Seite</a></li>
      <li><a href="drei.html">dritte Seite</a></li>
   </ul>
</div>
...
```

Im Rahmen der normalen CSS-Präsentation wird diese Sprungnavigation außerhalb des Sichtbereich (in der Regel links außen) positioniert. Sie ist damit im Webbrowser unsichtbar, im Bedarfsfall jedoch für Screenreader zugänglich.

```
#sprunglinks  {
   position: absolute;
   left: -10000px;
   height: 1px;
}
```

Ob Sie den Container mit den Sprunglinks absolut oder relativ positionieren, ist verhältnismäßig egal. Achten Sie bei relativer Positionierung jedoch darauf, die kleinstmögliche Höhe anzugeben, um Ihr Seitenlayout nicht zu verschieben.

22 Sie sollten dies grundsätzlich in Erwägung ziehen, da es die Barrierefreiheit des Dokuments in jedem Falle fördert.

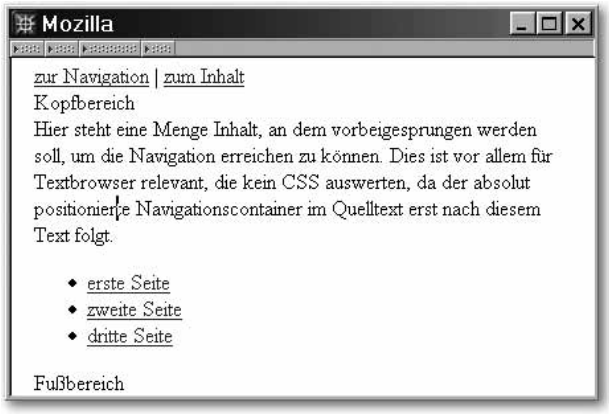

Abbildung 14.29 Anzeige der Sprunglinks ohne CSS

14.27 dl

Das Element `<dl>` kennzeichnet eine Definitionsliste (*definition list*), die abwechselnd zu definierende Begriffe und deren Definition enthält.

14.27.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	List
Kontext	Innerhalb von `<body>` und innerhalb von `<applet>`, `<blockquote>`, `< button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<map>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`
Erlaubte Inhalte	Elemente vom Typ `<dt>` und `<dd>`
Offizielle W3C Spezifikationen	HTML Level 0; HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<dl> ... </dl>
```

Defaultrendering:
Das Element wird standardmäßig als Block dargestellt.

CSS-Umsetzung des HTML-Defaults:

```
dl  { display:block; }
```

14.27.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`compact`
Internet Explorer	*Access-, Edit-Attribute*

14.27.3 Offizielle Attribute – W3C

compact

Wert	*"compact"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `compact`-Attribut reduziert den Abstand zwischen den Listenelementen der Liste und bewirkt eine kompaktere Darstellung der Liste.

Anmerkung: Das `compact`-Attribut wird in der Praxis von kaum einem Browser unterstützt. Es ist anzuraten, für die beabsichtigte Wirkung auf das CSS-`margin`-Property der Listenelemente zurückzugreifen.

14.27.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.27.5 Erläuterung zum Element

Eine Definitionsliste stellt zu definierende Ausdrücke und die dazugehörigen Definitionen als Liste dar. Für den Container sind keine Verkürzungen gestattet, es müssen also sowohl in XHTML als auch in HTML Start- und Endmarke gesetzt werden.

Als Inhalt sind in loser Abfolge `<dt>`- und `<dd>`-Container erlaubt, wobei jeweils mehrere Definitionsbegriffe oder mehrere Definitionen nacheinander stehen dürfen. Eine Definition kann sich also auf mehrere Definitionsterme beziehen, so wie analog für einen Begriff mehrere Definitionen zulässig sind.

Sowohl der `<dt>`-Container (*definition term*) als auch der `<dd>`-Container (*definition data*) stellen Blockelemente dar, werden also wie andere Listenelemente auch jeweils in einer neuen Zeile dargestellt, wobei die Definitionen eingerückt präsentiert werden.

In der Regel treten Definitionsterm und Definition paarweise auf, wobei die Definition auf den Begriff folgt:

```
<dl>
   <dt>Begriff</dt>
      <dd>Definition des Begriffs</dd>
...
</dl>
```

Das Inhaltsmodell der Definitionsliste erlaubt jedoch sehr wohl Folgen mehrerer `<dt>`- oder `<dd>`-Container innerhalb einer Liste. Es ist also möglich, alternative

Definitionen eines Begriffes oder alternative Begriffe zu einer Definition zu beschreiben:

```
<dl>
   <dt>Begriff</dt>
      <dd>Definition des Begriffs</dd>
      <dd>Alternative Definition des Begriffs</dd>
...
   <dt>Begriff</dt>
   <dt>Alternativer Begriff</dt>
      <dd>Definition des Begriffs</dd>
...
</dl>
```

Eine Verwendung des `<dd>`-Containers *ohne* vorangehenden Definitionsterm innerhalb einer Definitionsliste ist gemäß der DTD zwar erlaubt, jedoch semantisch nicht sinnvoll:

```
<dl>
   <dd>Dieser Text ist einger&uuml;ckt.</dd>
</dl>
```

Wegen der Präsentation der Definition mit Einrückung wurde dies gerne als Gestaltungselement eingesetzt. Dem steht jedoch die Richtlinien für Barrierefreiheit entgegen: Listenelemente dürfen nur für Listen und nicht beispielsweise für Texteinzüge im Fließtext verwendet werden (WAI 3.6, BITV 3.6).

[»] *Hinweis:* Die HTML-Spezifikation schlägt auch vor, Definitionslisten als Dialogmarkup zu nutzen (Beispieltext aus Shakespeare, Hamlet: 1. Aufzug, 1. Szene) – der Nutzung gemäß WAI würde dies jedoch wohl kaum entsprechen:

```
<dl>
<dt>Bernardo.</dt>
   <dd>Wer da?</dd>
<dt>Francisco. </dt>
   <dd>Nein, gebt Antwort: Halt, und sagt wer ihr seyd.</dd>
<dt>Bernardo. </dt>
   <dd>Lang lebe der K&ouml;nig!</dd>
...
</dl>
```

[zB] *Beispiel:*

```
<dl>
<dt>Gef&auml;hrlich:</dt>
<dd>Einen Boxer reizen</dd>
<dt>Sehr gef&auml;hrlich:</dt>
<dd>Zwei Boxer reizen</dd>
<dt>Höchst gef&auml;hrlich:</dt>
```

```
<dd>Einen Pudel reizen</dd>
</dl>
```

Abbildung 14.30 Eine Definitionsliste <dl> mit Definitionen <dt> und <dd>

Bildunterschriften mit Definitionslisten

Ein interessanter Einsatz für Definitionslisten (in semantischem Sinne vielleicht gar nicht einmal eine Zweckentfremdung) bietet sich im Zusammenhang mit der Notwendigkeit, Bilder und Bildunterschriften zusammenzufassen.

Hierzu schließt man Bild und Bildunterschrift in einen `<dl>`-Container ein, wobei das Bild die Rolle des Definitionsbegriffs und die Bildunterschrift die der Definition übernimmt. Die Liste wird mittels einer CSS-Klasse erfasst:

```
<dl class="bildcontainer">
   <dt><img src="mein_bild.jpg" alt="Mein Bild" /></dt>
   <dd>Die Bildunterschrift</dd>
</dl>
```

Gibt man dem Listencontainer über CSS eine zum Bild passende Breite und setzt die Margins des `<dd>`-Containers auf Null (oder einen anderen geeigneten Wert). Der gesamte Container kann zusätzlich gefloatet werden, um das Bild mit zugehöriger Untertitelung innerhalb eines Fließtextes zu platzieren (Abbildung 14.31).

Abbildung 14.31 Bild mit Untertitel in <dl>-Container gruppiert

Die für Abbildung 14.31 eingesetzten CSS-Präsentationen (die Containerbreite ist an das im Beispiel verwendete Bild angepasst – der Rahmen dient der Verdeutlichung im Screenshot):

```
dl.bildcontainer    { width:130px;
                      padding:0.5em;
                      margin:0.5em;
                      border:1px dotted black;
                      float:left;
}

dl.bildcontainer  dd  { margin:0;
                        font-style:italic;
}
```

Die Präsentation kann beliebig verfeinert werden, indem Sie Inhalte zentrieren oder zusätzlich gezielt Paddings, Margins und Hintergrundbilder einsetzen.

14.28 dt

Das Element `<dt>` kennzeichnet den zu definierenden Begriff (*definition term*) innerhalb einer Definitionsliste.

14.28.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Container
XHTML-Modul	List
Kontext	Innerhalb einer Definitionsliste `<dl>`
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<dt> ... </dt>
```

Defaultrendering:
Das Element wird standardmäßig als Block mit einer Einrückung dargestellt.

CSS-Umsetzung des HTML-Defaults:

```
dt  { display:block; margin-left: 40px}
```

14.28.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	nowrap, *Access-, Edit-Attribute*

14.28.3 Proprietäre Attribute – Internet Explorer

nowrap

Wert	*"nowrap"* | *"true"* | *"false"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	IE3

Das nowrap-Attribut erlaubt bzw. unterdrückt den automatischen Zeilenumbruch innerhalb des Elements. Defaultmäßig bricht der Browser lange Zeilen automatisch um. Der Wert nowrap="false" gestattet ihm dies (überflüssigerweise) explizit, während nowrap="true" bzw. nowrap="nowrap" den Umbruch verbieten. Gegebenenfalls erscheint hierdurch eine horizontale Scrollleiste.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.28.4 Erläuterung zum Element

Der `<dt>`-Container kennzeichnet innerhalb einer Definitionsliste den zu definierenden Begriff. In der Regel wird er zusammen mit einer oder mehreren folgenden Definitionen `<dd>` eingesetzt.

Als Inhalt des Containers sind nur Inlineelemente zugelassen, was aber auch Grafiken, Objekte und Formularelemente (nicht den `<form>`-Container selbst) umfasst.

Dies ermöglicht es, Definitionslisten (hier über eine CSS-Klasse gekennzeichnet) auch einzusetzen, um Formularfelder und zu ihnen gehörende Erläuterungstexte zusammenzufassen (siehe Abbildung 14.32):

```
<form action="beispiel.cgi">
<p><strong>Definitionsliste für Felderl&auml;uterungen in
Formularen:</strong></p>
<dl class="felddefinition">
   <dt>
      <label for="vn"><strong>Vorname(n):</strong></label>
      <input type="text" id="vn"/>
   </dt>
   <dd>
      <em>Erl&auml;uterung:</em> Geben Sie hier
       alle Ihre Vornamen ein.
   </dd>
   <dt>
   <label for="nn"><strong>Nachname:</strong></label>
   <input type="text" id="nn"/>
   </dt>
   <dd>
```

```
      <em>Erl&auml;uterung:</em> Geben Sie hier
       Ihren Nachnamen ein.
   </dd>
</dl>
</form>
```

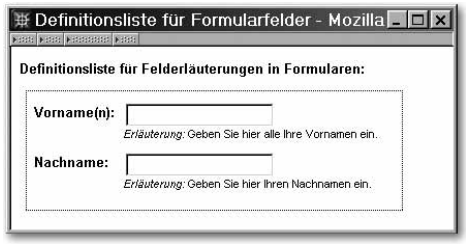

Abbildung 14.32 Definitionsliste gruppiert Formularfelder und Erläuterungen

Das Vorgehen ist analog zu dem im Vorfeld beschriebenen Verfahren, Definitionslisten für Bildunterschriften einzusetzen.

In diesem Beispiel wird das `<label>`-Element per CSS mit einer Breitenangabe versehen, um die Eingabefelder auszurichten. Für den `<dd>`-Erläuterungstext ist ein entsprechender linker Rand festgelegt (unten fett markiert):

```
label                  { display:block;
                         float:left;
                         width:7em;
}

dl.felddefinition      { width:350px;
                         padding:1em 0.5em 0.5em 0.5em;
                         margin:0.5em;
                         border:1px dotted black;
}

dl.felddefinition  dd  { margin:0 0 1em 7.5em;
                         font-size:90%;
}
```

Querverweis: Weitere Informationen zu Definitionslisten finden Sie bei der ausführlichen Beschreibung des `<dl>`-Containers.

14.29 em

Logische Auszeichnung: Der `<em>`-Container markiert seinen Inhalt als »betont« (*emphasized*) im Sinne einer Hervorhebung.

14.29.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Innerhalb von Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<em> ... </em>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
em     { font-style: italic }
```

14.29.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.29.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.29.4 Erläuterung zum Element

Das Element `<em>` dient der logischen Hervorhebung als »betont«. Es wird gewöhnlich kursiv präsentiert und stellt hiermit das Pendant zur rein präsentativen Kursivauszeichnung durch `<i>` dar. Bei `<em>` handelt es sich im Gegensatz zu `<i>` jedoch um eine semantische, bedeutungstragende Auszeichnung. Das Element ist Teil aller HTML- und XHTML-Vokabulare.

Ein Screenreader wird einen mit `<em>` markierten Textabschnitt als »betont« erkennen und mit entsprechend erhöhter Lautstärke vortragen. Das Verhalten eines Screenreaders gegenüber einem Italic-Container `<i>` ist hingegen undefiniert, da keine Aussage über die Bedeutung des Textinhalts gemacht wird. Ein Screenreader wird `<i>` also ebenso wie `<b>` (»bold«) ignorieren, da es sich bei beiden Elementen um rein optische Auszeichnungen handelt.

Die optische Präsentation ist bei `<em>` von der logischen Bedeutung unabhängig – obwohl es eine Defaultpräsentationen als »kursiv« gibt, spricht nichts dagegen, das Element beispielsweise mit Fettauszeichnung (oder farblicher Hervorhebung) darzustellen.

```
/* <em> in nicht-kursiver, roter Schrift: */
em { font-style:normal; color:red; }
```

Entscheidend ist die semantische Bedeutung des Elements, nicht seine Darstellung. Eine ähnliche Umwidmung (z. B. <i> als fett dargestellt) hätte bei den Präsentationselementen selbstverständlich keinen Sinn.

Praxistipp

Verwenden Sie die logischen Auszeichnungen und , wenn Sie verschiedene Abstufungen semantischer Hervorhebungen (»Wichtigkeit«) in Ihrem Text markieren wollen. Vermeiden Sie den Einsatz von <i> und <b> für diesen Zweck – als Präsentationselemente sind Letztere auch nicht mehr Teil der *strict*-DTDs und daher nicht zukunftstauglich.

[zB] *Beispiel:*

```
<p>Ich m&ouml;chte <em>wirklich</em> gern HTML lernen!</p>
<p>Alle sagen, das sei so
 <em title="Wer&apos;s glaubt!">einfach</em>!</p>
```

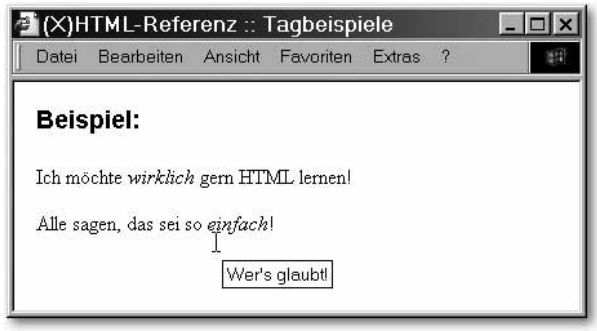

Abbildung 14.33 Betonung mit markiert; unten mit Tooltip durch title-Attribut

14.30 embed

Nicht spezifiziert. Das Element <embed> dient der Einbettung beliebiger Medienobjekte in die HTML-Seite.

14.30.1 Elementeigenschaften

Einführung	Nicht offiziell spezifiziert (Netscape 2)
Elementtyp	Inlineelement, leer; Endmarke jedoch erlaubt
XHTML-Modul	–
Kontext	In allen Block und Inlineelementen (meist als Fallbacklösung in <object> eingesetzt)
Erlaubte Inhalte	Keine (leeres Element)

Offizielle W3C Spezifikationen	Nicht spezifiziert
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

`<embed [Attribute] >` [`</embed>`] (HTML-Kontext)

`<embed [Attribute] />` (XHTML-Kontext)

14.30.2 Attribute (Übersicht)

Standardattribute

HTML	–
XHTML	–

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

allgemein gültig (nicht offiziell spezifiziert)	`align`, `alt`, `autostart`, `border`, `code`, `codebase`, `frameborder`, `height`, `hidden`, `hspace`, `loop`, `name`, `pallette`, `playcount`, `pluginspage`, `pluginurl`, `src`, `type`, `units`, `vspace`, `volume`, `width` (und weitere Attribute, abhängig vom Plug-in-Typ des Objekts)
Web Application 1.0	`height`, `src`, `type`, `width`

14.30.3 Allgemein verwendbare Attribute

align

Wert	*"top"* | *"middle"* | *"bottom"* | *"left"* (float) | *"right"* (float)
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut erlaubt es, ein eingebettetes Objekt so wie ein Bild im Zeilenkontext oder mittels Float `"left"` oder `"right"` am entsprechenden Rand des übergeordneten Containers zu positionieren.

alt

Wert	*cdata*
Typ	Alternative Textbeschreibung
Verwendung	Optional

Das `alt`-Attribut bindet einen Alternativtext an das eingebettete Objekt, der verwendet wird, falls die Einbettung nicht möglich ist.

autostart

Wert	*"true"* | *"false"*
Typ	Boolesche Bezeichner
Verwendung	Optional

Das `autostart`-Attribut bestimmt, ob die Wiedergabe des eingebetteten Objekts automatisch starten soll (*true*) oder nicht (*false*). Der Defaultwert ist uneinheitlich implementiert.

border

Wert	*number*
Typ	Ein numerischer Wert, als Pixel interpretiert
Verwendung	Optional

Das `border`-Attribut bestimmt die Breite des Rahmens um das eingebettete Objekt in Pixel.

code

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `code` nennt den absoluten oder relativen URL der in Form einer Einbettung zu ladenden Javaklasse mit der Dateiendung `.class`. Das Attribut ist *case-sensitive*.

codebase

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `codebase`-Attribut gibt einen absoluten oder relativen URL an, der als Basis-URL für den URL des `code`-Attributs dient. Ein relativer URL wird als bezogen auf den URL des Dokuments oder, falls ein solcher vorhanden ist, auf den `href`-Wert des `<base>`-Tags ausgewertet.

frameborder

Wert	*"0"* | *"1"*
Typ	Boolesche Werte in Form der Ganzzahlen 0 und 1
Verwendung	Optional

Das `frameborder`-Attribut bestimmt, ob ein Rahmen um das Objekt gezeigt wird (`frameborder="1"`, Default) oder nicht (`frameborder="0"`, als »borderless plugin«).

height

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `height`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für die Höhe des für das Applet freizuhaltenden Raums.

hidden

Wert	*"true"* | *"false"* (Default)
Typ	Boolesche Bezeichner
Verwendung	Optional

Das Attribut `hidden` bestimmt, ob das eingebettete Objekt dargestellt werden soll (`hidden="false"`, Default) oder verborgen werden soll (`hidden="true"`). Letzteres ist beispielsweise für eingebettete Audiodateien sinnvoll.

hspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `hspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den links und rechts vom eingebetteten Objekt freizuhaltenden Raum.

loop

Wert	*"true"* | *"false"* | *[integerwert]*
Typ	Boolesche Bezeichner oder Ganzzahl
Verwendung	Optional

(Netscape) Das `loop`-Attribut bestimmt, ob die Wiedergabe des eingebetteten Objekts endlos (*true*) oder nur einmal erfolgen soll (*false*). Alternativ kann mit einem Ganzzahlwert die Zahl der Wiederholungen festgelegt werden.

name

Wert	*nmtoken*
Typ	Zeichenkette in Form eines Nametokens
Verwendung	Optional

Das `name`-Attribut nennt eine Zeichenkette, die von Scripten als Bezeichner für das eingebettete Objekt verwendet werden kann. Alternativ kann das `id`-Attribut eingesetzt werden. Das Attribut ist *case-sensitive*.

[»] *Hinweis:* Wenn Sie ein `<embed>` in einen `<object>`-Container einbetten, so verwenden Sie *keinesfalls* für beide Elemente den gleichen Namen. Sie können jedoch problemlos für `<object>` einen zum `name`-Attribut des inneren `<embed>` gleichlautendes `id`-Attribut einsetzen.

palette

Wert	*"foreground"* | *"background"*
Typ	Farbwahl zur Darestellung des Objektbereichs
Verwendung	Optional

Mit dem `palette`-Attribut kann festgelegt werden, ob der Bereich des eingebetteten Objekts in der Schriftfarbe (`palette="foreground"`) oder der Farbe des Dokumenthintergrundes (`palette="background"`) dargestellt werden soll.

playcount

Wert	*integerwert*
Typ	Ganzzahl
Verwendung	Optional

(Internet Explorer) Das `playcount`-Attribut nennt die Anzahl der Wiederholungen, wenn mittels `<embed>` ein Audio- oder MIDI-File eingebettet wird.

pluginspage

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `pluginspage`-Attribut nennt die Adresse derjenigen Webseite (*page*), auf der das für das Objekt benötigte Plug-in zur Verfügung gestellt wird.

Hinweis: Das Attribut wird oft fälschlich `pluginspace` geschrieben. **[«]**

pluginurl

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

(Netscape) Das Attribut `pluginurl` nennt ein JAR-Archiv (in Netscape) als Ressource zur automatischen Installation des für das eingebettete Objekt benötigten Plug-ins. In Mozilla (Gecko Engine) zeigt das Attribut auf ein so genanntes »XPInstall file« vom Typ `.xpi` (im Prinzip ein Zip-Wrapper für `.exe`- und weitere benötigte Dateien, die der Plug-in-Hersteller zur Verfügung stellt).

src

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `src`-Attribut legt einen URL für den Download des Objekts fest.

type

Wert	*cdata*
Typ	Ein MIME-Typ-Bezeichner
Verwendung	Optional

Das `type`-Attribut nennt den MIME-Typ der zu ladenden Ressource.

units

Wert	*"px" | "em" | [CSS-Maßeinheit]*
Typ	Bezeichner von Längeneinheiten
Verwendung	Optional

(Internet Explorer) Der String des `units`-Attributs bestimmt, ob die Maßangaben des eingebetteten Objekts als Pixelwerte (Default), in em-Breiten oder einer anderen gültigen CSS-Maßeinheit zu interpretieren sind.

vspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `vspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch über und unter dem Applet freizuhaltenden Raum.

volume

Wert	*integerwert*
Typ	Ganzzahl zwischen 0 und 100 (Default: 50)
Verwendung	Optional

(Netscape) Das `volume`-Attribut steuert die Wiedergabelautstärke eines durch `<embed>` eingebetteten Audiofiles. Erlaubte Werte liegen zwischen 0 (minimale Lautstärke) und 100 (maximale Lautstärke). Die exakte Implementierung ist abhängig von Browser und Betriebssystem.

width

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `width`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für die Breite des für das Applet zu reservierenden Raums.

14.30.4 Erläuterung zum Element

Das `<embed>`-Element wurde von Netscape initiiert, um Einbettung und Wiedergabe von Multimediainhalten in HTML-Seiten zu ermöglichen. Im Gegensatz zum gleichzeitig eingeführten `<applet>`-Element wurde `<embed>` nachfolgend nie standardisiert, wird aber noch breit unterstützt (Internet Explorer, Mozilla/Firefox, Opera). Da `<embed>` kein offizieller Bestandteil von HTML ist, sind Dokumente, in denen es eingesetzt wird, jedoch nicht erfolgreich validierbar.

Prinzipiell darf der `<embed>`-Container leer bleiben, aber auch Text, Inline- und Blockelemente enthalten. Eine verkürzte Schreibweise eines leeren Containers gemäß XHTML-Syntax als `<embed />` ist nicht ratsam. Setzen Sie also stets eine Endmarke.

Standardisierte Alternative: <object>

Verwenden Sie `<embed>` nach Möglichkeit nicht, sondern weichen Sie auf das standardisierte `<object>`-Element aus!

Wildwuchs an Attributen

Neben den vorgestellten Attributen existieren noch weitere, da `<embed>` als offenes Konzept zur Einbettung beliebiger Medientypen angelegt wurde, das von Plug-in-Herstellern beliebig erweitert werden konnte (dies war sicher Ursache für die Nichtstandardisierung des Elements).

Dass die Attribute nicht nur vom Browser selbst, sondern auch vom installierten Plug-in (als so genannte *private attributes*) ausgewertet werden, führt(e) zu entsprechendem Wildwuchs. Zu nennen sind Attribute wie `controls`, `mastersound`, `endtime`, `starttime` (alle Netscape), `controller`, `href`, `target` (Internet Explorer) oder `autohref`, `cache`, `kioskmode` und etwa zwanzig weitere, die für Apple Quicktime[23] eingeführt wurden.

Anmerkung: Bei `<object>` ist die Weitergabe beliebig benennbarer Datenobjekte durch das `<param>`-Element gelöst, um es Plug-in-Herstellern zu ersparen, eigens Attribute einführen zu müssen.

23 Eine Übersicht erhalten Sie unter www.apple.com/quicktime/tutorials/embed.html.

Problem WYSIWYG-Editoren

Als problematisch ist in diesem Zusammenhang zu werten, dass viele Webeditoren `<embed>` noch immer automatisch als Fallback für mittels `<object>` eingebettete Flashfilme einsetzen und so die Validierbarkeit erzeugter Dokumente unterlaufen.

Das folgende Quellcodebeispiel wurde von Dreamweaver erzeugt (Plug-in-URLs aus Gründen der Übersichtlichkeit gekürzt):

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/..."
 width="100" height="100">
<param name=movie value="intro.swf"/>
<param name=quality value=high/>

   <embed src="intro.swf" quality=high
pluginspage="http://www.macromedia.com/shockwave/..."
    type="application/x-shockwave-flash"
    width="100" height="100"></embed>

</object>
```

Hintergedanke ist eine erfolgreichere Einbettung des Flash-Objekts, daher wird dieses Verfahren von Macromedia (jetzt Adobe) noch immer empfohlen. Die Einbindung durch `<object>` erfolgt nach Microsoft-Vorgaben (daher das `classid`- und das `codebase`-Attribut. Das innere `<embed>` spiegelt hingegen die Vorgehensweise von Netscape wider.

Eine standardkonforme Einbettung muss auf `<embed>` verzichten und sieht demzufolge so aus (beachten Sie die Hinzunahme der Attribute `data` und `type`):

```
<object data="intro.swf" width="100" height="100"
 type="application/x-shockwave-flash">
   <param name="movie" value="intro.swf"/>
   <param name="quality" value="high"/>
</object>
```

Alternativinhalt mit <noembed>

Ebenso inoffiziell wie `<embed>` selbst ist der Container `<noembed>`, der Alternativinhalte zu den eingebetteten Objekten zur Verfügung stellen soll. Das Prinzip besteht einfach darin, dass diejenigen Browser, die `<embed>` beachten, den `<noembed>`-Container mitsamt Inhalt ignorieren, andere dagegen sowohl die Marken für `<embed>` als auch für `<noembed>` verwerfen, den Inhalt des `<noembed>`-Containers jedoch darstellen:

```
<embed src="/sounds/1812over.mid" autostart="true"></embed>
<noembed>
   <p>Hier k&ouml;nnten Sie Tschaikowsky h&ouml;ren...</p>
</noembed>
```

Die Zukunft des <embed>-Tags

Derzeit existiert das Element offiziell nicht, was sich jedoch ändern mag: In der durch die WHAT WG propagierten Spezifikation zu »Web Applications 1.0« (populär als »HTML 5« bezeichnet) wird vorgeschlagen, das `<embed>`-Element letztendlich doch als Standard einzuführen[24].

Vorgesehen sind als elementspezifische Attribute dabei lediglich `src` und `type` (beide obligatorisch) sowie `height` und `width` (beide optional). Das Element soll jedoch ausdrücklich als leeres Element realisiert werden und keinen »Fallback-Content« besitzen. Ob ebenfalls ein `<noembed>`-Element spezifiziert werden soll, geht aus den Vorschlägen nicht eindeutig hervor (könnte jedoch abgeleitet werden).

14.31 fieldset

Der `<fieldset>`-Container markiert eine in ihm eingeschlossene Gruppe von Inputelementen (*fields*) eines Formulars als logisch zusammengehörigen Set.

14.31.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Blockcontainer
XHTML-Modul	Forms
Kontext	Beliebig innerhalb eines `<form>`-Containers (darf dabei auch verschachtelt werden)
Erlaubte Inhalte	Elemente von Typ `<legend>`, außerdem Block- und Inlineelemente und Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1; **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile **abgeschafft** in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<fieldset> ... </fieldset>
```

24 Siehe http://whatwg.org/specs/web-apps/current-work/#the-embed.

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
fieldset    { display: block;
              margin: 1.33em 0;
              border: 2px groove #000;
             }
```

14.31.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onafterupdate, onbeforecopy, onbeforecut, onbeforeeditfocus, onbeforepaste, onbeforeupdate, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfilterchange, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.31.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false"` | `"true"`

`unselectable = "on" | "off"` (IE 5)

14.31.4 Erläuterung zum Element

Mit Hilfe des Blockcontainers `<fieldset>` können logisch zusammengehörende Eingabeelemente eines Formulars zu einer Gruppe zusammengefasst werden. Eine Auswirkung auf die Funktion der Inputs hat der Container nicht. Optisch wird die die Zusammengehörigkeit der eingeschlossenen Elemente durch einen umlaufenden Rahmen hervorgehoben, in den eine mit `<legend>` definierte Gruppenüberschrift eingegliedert werden kann.

Inhalte von Fieldsets

Das `<fieldset>`-Element darf laut Spezifikation neben Flow-Elementen (d.h. Block- und Inlineelemente) auch `#PCDATA` enthalten, wobei vor dem optionalen `<legend>`-Element ausdrücklich nur Whitespace-Zeichen auftreten dürfen.

In der Praxis ist dies aufgrund des liberalen Inhaltsmodells nicht reglementierbar. Abweichungen hiervon können folglich durch eine Validierung des Dokuments nicht festgestellt werden.

```
<!-- illegal, aber validierbar: -->
<fieldset>
   Hier nur Whitespace erlaubt!
   <legend>Test</legend>
   Hier ist Text gestattet.
</fieldset>
```

Das vorstehende Quellcode-Fragment wird browserabhängig verschieden interpretiert. Der Internet Explorer schlägt einem vor dem `<legend>`-Container stehenden Text zwar dem `<fieldset>`-Block zu, zeigt ihn aber außerhalb von dessen Rahmen. Textinhalt, der auf den `<legend>`-Container folgt, liegt in jedem Fall innerhalb des Rahmens.

Die restlichen aktuellen Browser stellen sämtliche im `<fieldset>`-Container enthaltenen Inhalte im Inneren des `<fieldset>`-Bereichs dar (es wäre eine Diskussion wert, welches Verhalten im Grunde das korrektere ist). Beachten Sie auch das Problem, das Opera bei vorangestelltem Text mit der Position der Legende hat (Abbildung 14.34):

Abbildung 14.34 #PCDATA-Problem bei <fieldset>-Inhalt

Eine **Verschachtelung** von `<fieldset>`-Elementen ist gestattet, so dass auch Feldgruppen wiederum gruppiert und mit Überschriften versehen werden können. Hierbei entsteht durch den äußeren `<fieldset>`-Container ein Rahmen um innere Gruppen. Hier besitzen diese ebenfalls eigene Überschriften:

```
...
<fieldset><legend>Personendaten</legend>
   <fieldset><legend>Vor- und Nachname</legend>
   ... <!-- Eingabefelder -->
   </fieldset>
   <fieldset><legend>Postanschrift</legend>
   ... <!-- Eingabefelder -->
   </fieldset>
   ...
</fieldset>
...
```

Fieldset zur Textpräsentation

Als Blockelement ist der `<fieldset>`-Container laut DTD auch als unmittelbarer Inhalt von `<body>` gestattet. Es ist daher im Prinzip legal, das `<fieldset>`-Element außerhalb eines Formulars in rein präsentativer Form zur Hervorhebung von Text einzusetzen. Streng genommen ist dies jedoch nicht im Sinne der Bedeutung des Elements. Bei einer Validierung bleibt folgender »Missbrauch« unbeanstandet:

```
<fieldset>Dieser Text wird umrahmt.</fieldset>
```

[zB] *Beispiel:* In einem Formular werden mit `<fieldset>`-Containern zwei Gruppen von Eingabeelementen gebildet und die erste wird mit einer Überschrift versehen:

```
<form action="beispiel.cgi">
<p>W&auml;hlen Sie hier die Art des Blumenstrau&szlig;es:</p>
<fieldset><legend>Blumen</legend>
<input name="blume" type="radio"/>Rosen
<input name="blume" type="radio"/>Tulpen
<input name="blume" type="radio"/>Nelken
</fieldset>
<p>W&auml;hlen Sie hier die Art der Zahlung:</p>
<fieldset>
<input name="zahlung" type="radio"/>Scheck
<input name="zahlung" type="radio"/>Karte
<input name="zahlung" type="radio"/>bar
</fieldset>
 ...
</form>
```

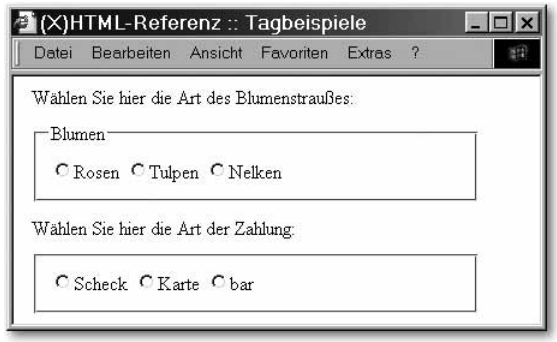

Abbildung 14.35 Fieldset mit und ohne Legende

14.32 font

Deprecated: Der `<font>`-Container ist ein reines Präsentationselement, das Schriftart, -größe und -farbe des eingeschlossenen Textes festlegt.

14.32.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Inlinecontainer
XHTML-Modul	Legacy
Kontext	Unmittelbar in allen Block und Inlineelementen mit Ausnahme von `<body>` und `<pre>`
Erlaubte Inhalte	Inlineelemente (auch `<font>` selbst) und Text (`#PCDATA`)

Offizielle W3C Spezifikationen	HTML 3.2, 4.0/4.01 (tr/fs), XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0/4.01 strict, XHTML 1.0 strict, 1.1, 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<font [Attribute]> ... </font>
```

Defaultrendering:
Der `<font>`-Container selbst ist darstellungsneutral – die gewünschte Präsentation wird über seine Attribute festgelegt.

CSS-Umsetzung des HTML-Defaults:

```
font { display:inline; }
```

14.32.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`color`, `face`, `size`
Internet Explorer	*Access-, Edit-Attribute*
Netscape	`point-size`, `font-weight`* *abgeschafft
Web-TV	`effect`, `transparency`

14.32.3 Offizielle Attribute – W3C

color

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Ein Farbname oder ein RGB-Wert in Hexadezimalschreibweise *#rrggbb*. Das Attribut `color` bestimmt die Textfarbe innerhalb des `<font>`-Containers. In Abwesenheit des Attributs gilt die im Body vereinbarte Textfarbe, ansonsten die Defaulttextfarbe `#000000`, Schwarz, oder die User-Einstellung des Browsers.

CSS-Alternative: `color:#rrggbb;`

face

Wert	*fontname_1 [, fontname_2, ... fontname_N]*
Typ	Kommagetrennte Liste aus einem bis N Fontbezeichnern
Verwendung	Optional

Das Attribut `face` nennt den für die eingeschlossenen Textabschnitte zu wählenden Font und, in der übergebenen Liste auf diesen folgend, alternative Fonts, auf die der Reihenfolge nach zurückgegriffen werden soll. Es ist sinnvoll, hier Schriften gleicher Gattung anzugeben und von der exakten Forderung über die Alternativen zur allgemeinen Gattung abzustufen, etwa:

```
face = "arial, helvetica, sans-serif"
```

Die Gültigkeit des `face`-Attributs erstreckt sich nur auf das Innere des aktuellen `<font>`-Containers und vererbt sich auch auf in diesem enthaltene `<font>`-Container, sofern in diesen kein `face`-Attribut gesetzt ist.

CSS-Alternative: `font-family: fontbezeichner;`

size

Wert	*n* | *+n* | *-n*
Typ	Numerische Angabe; ggfs mit Vorzeichen + oder –
Verwendung	Optional

Die Gültigkeit der durch das `size`-Attribut vorgenommenen Schriftgrößenangabe erstreckt sich auf das Innere des aktuellen `<font>`-Containers und vererbt sich auch auf in diesem enthaltene `<font>`-Container, sofern in diesen kein `size`-

Attribut gesetzt ist. In Abwesenheit des Attributs gilt für normale Textabschnitte (nicht für Überschriften) die Defaultschriftgröße 3.

Die Größe kann entweder in den absoluten Größenangaben 1 (sehr klein) bis 7 (sehr groß) oder in relativen Angaben +1 bis +7 (größer) bzw. –1 bis –7 (kleiner) festgelegt werden, die sich auf die Ausgangsschriftgröße 3 beziehen oder auf die eines übergeordneten `<font>`-Containers.

[!] *Achtung:* Überschriften `<h1>` bis `<h6>` sind durch das `size`-Attribut unterschiedlich betroffen – je nach Hierarchiestufe der Überschrift können in der Summe mit den Größenangaben durch `<font>`-Container die Extremwerte Größe 7 und Größe 1 nicht über- bzw. unterschritten werden.

CSS-Alternative: `font-size: number[px][%] | bezeichner;`

14.32.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.32.5 Proprietäre Attribute – Netscape

point-size

Wert	*number*
Typ	Eine positive Ganzzahl
Verwendung	Optional
Einführung	NN4; unterstützt in Mozilla/Firefox

Das `point-size`-Attribut erlaubt die exakte Größenangabe der Schrift in Punkt, wobei die durch das `size`-Attribut ermöglichten Größen auch unter- bzw. überschritten werden können.

```
<font point-size="65">point-size!</font>
```

Abbildung 14.36 Das point-size-Attribut in Netscape 4

Anmerkung: Das Attribut wird gegenwärtig in Mozilla/Firefox noch unterstützt.

font-weight

Wert	*number*
Typ	Eine positive Ganzzahl
Verwendung	Optional
Einführung	NN4

(Historisch) Ein numerischer Wert für den Fettegrad von 100 (leicht) bis 900 (sehr fett) oder eine vorzeichenbehaftete, relative Angabe. Das `font-weight`-Attribut entspricht in seinem Verhalten dem gleichnamigen CSS-Property. Der Wert 400 entspricht »normal«. Das Attribut wird in aktuellen Browsern nicht mehr unterstützt. (Anmerkung: In verschiedenen älteren Netscape-Dokumentationen wird das Attribut fälschlich mit `weight` bezeichnet.)

14.32.6 Proprietäre Attribute – WebTV

effect

Wert	*"relief"* | *"emboss"* | *"shadow"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `effect`-Attribut kann in WebTV-Anwendungen eingesetzt werden, um auf die eingeschlossenen Textpassagen Relief-, Embossing- oder Schatteneffekte anzuwenden, sollte jedoch heutzutage durch entsprechende CSS-Angaben ersetzt

werden. (Für Schatteneffekte kann alternativ das proprietäre WebTV-Element `<shadow>` eingesetzt werden.)

transparency

Wert	*number (Angabe 0 – 100)*
Typ	Ganzzahl
Verwendung	Optional

Mittels des `transparency`-Attributs kann in WebTV-Anwendungen die Transparenz der eingeschlossenen Textpassagen gesteuert werden. Der Wertebereich reicht von 0 (vollkommen opak, Default) bis 100 (vollkommen transparent).

14.32.7 Erläuterung zum Element

Der `<font>`-Container ist noch Teil der *transitional*- und *frameset*-DTDs von HTML 4.01 und XHTML 1.0 und als solcher durchaus noch ein gültiges Element. Da er jedoch rein präsentativen Charakter hat und sich obendrein ausschließlich an visuell orientierte User Agents richtet, hat er im zeitgenössischen Webdesign keine Daseinsberechtigung mehr.

Auch WAI und BITV betonen, dass Schriftformate keinesfalls mehr mit `<font>`-Elementen, sondern mit CSS-Properties anzugeben seien, damit sehbehinderte Menschen die Darstellung der Schriftgröße in ihrem Browser individuell einstellen können (siehe WAI 3.3, BITV 3.4). Das Verzichten auf den `<font>`-Container oder seine Eliminierung in älteren Dokumenten stellt also einen positiven Schritt in Richtung Barrierefreiheit dar.

Unnötige Dateigröße

Die Verwendung von `<font>` bläht die Dateigröße Ihres HTML-Dokuments unnötig auf und verringert die Lesbarkeit des Quellcodes. Der `<font>`-Container ist ein Inlinecontainer, was die Wiederholung immer gleicher Darstellungsanweisungen innerhalb aller Blockcontainer erfordert. Da Inlineelemente nicht außerhalb von Blockelementen auftreten dürfen, ist ein gleichzeitiges Anwenden auf mehrere Textabsätze oder eine gesamte Tabelle durch einen umgebenden `<font>`-Container verboten. Das Element muss(te) also in alle Tabellenzellen, Textabsätze, Überschriften, Listenelemente und so fort einzeln eingefügt werden.

Signifikant kleiner ohne <font>

Eine Verringerung der Dateigröße eines HTML-Dokument um bis zu 30 % durch das Entfernen von `<font>`-Elementen ist nicht unrealistisch.

Fehlende semantische Bedeutung

Der `<font>`-Container trifft keinerlei Aussage über die logische Bedeutung seines Inhaltes innerhalb der Dokumentstruktur.

- Hervorhebungen (Warnungen, Schlüsselbegriffe) lassen sich besser mit logischen Elementen wie `<em>` oder `<strong>` vornehmen, gegebenenfalls unter Zuhilfenahme von CSS-Klassen. Alternativ können Sie den neutralen Inlinecontainer `<span>` in Verbindung mit einer Stylesheetklasse verwenden.
- Mit `<font>` vergrößerter oder fett gesetzter Text, der im Sinne einer Überschrift eingesetzt wird, sollte korrekter mit einem `<h...>`-Container realisiert werden.

Ersatz durch CSS-Präsentationen

Sie können sämtliche Präsentationswirkungen der HTML-Attribute von `<font>` ebenso gut und besser mit CSS vornehmen.

HTML-Attribut	Wert	CSS-Property	Wert
`color`	Farbname RGB-Hexwert	`color`	Farbname RGB-Hexwert RGB-Dezimalwert
`size`	absolut numerisch relativ numerisch (mit Vorzeichen)	`font-size`	Keyword beliebige absolute oder relative Angabe
`face`	Fontnamen (Liste)	`font-family`	Fontnamen (Liste)

Umbau älterer Dokumente

Wenn Sie ein vor einiger Zeit erstelltes HTML-Dokument überarbeiten, stellen Sie vermutlich fest, dass es eine Reihe von `<font>`-Anweisungen enthält, die Sie nun lieber loswerden möchten. Sie können hierfür HTML Tidy einsetzen, wie es in Kapitel 8, *Aktualisierung alter Websites,* beschrieben ist. Folgende Vorarbeiten sind jedoch sinnvoll:

1. Stellen Sie zunächst fest, welche Schriftart die im Dokument vorherrschende ist, und machen Sie diese zur Defaultschrift, indem Sie sie mit CSS-`font-family` beispielsweise dem `<body>`-Container zuweisen.

2. Erstellen Sie Stylesheetklassen für jene Fälle, wo <font> für weitere Präsentationen verwendet wird, die Sie beibehalten wollen oder müssen (Farben und Schriftgrößen oder punktuelle Änderungen der Schriftart). Verwenden Sie sprechende Klassenbezeichner, die die (gemeinte) logische Bedeutung der Formatierung widerspiegelt, sofern dies möglich ist.
3. Entfernen Sie nun alle <font>-Elemente und ersetzen Sie diese durch die vorbereiteten CSS-Angaben.

Tipp für Dreamweaver-Nutzer

Falls Sie Dreamweaver MX einsetzen (oder eine frühere Version des Programms), leistet Ihnen dessen Funktion **Strip Tag** beim globalen Entfernen von <font>-Tags gute Dienste.

[zB] *Beispiel:*

```
<font size="7">Gr&ouml;&szlig;e: size="7"</font><br>
<font size="6">Gr&ouml;&szlig;e: size="6"</font><br>
<font size="5">Gr&ouml;&szlig;e: size="5"</font><br>
<font face="Arial, Helvetica, sans-serif" size="1">
Gr&ouml;&szlig;e, Face: face="Arial, Helvetica, sans-serif" size="1"
</font><br>
<font color="#ff0000">Farbe: color="#ff0000"</font><br>
<font face="courier">Face: face="courier"</font>
```

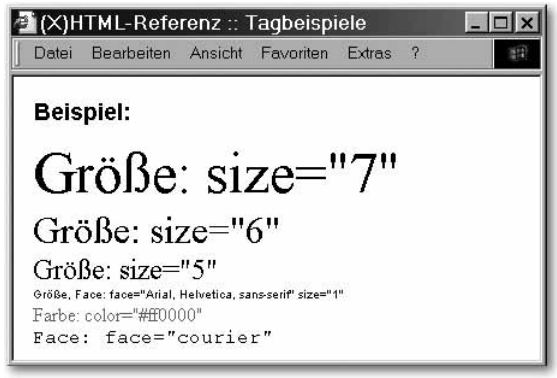

Abbildung 14.37 Verschiedene, mit <font> bestimmte Schriftgrößen

14.33 form

Der <form>-Container fasst alle Formularelemente (Inputs) in seinem Inhaltsbereich zu einem gemeinsam arbeitenden Webformular zusammen.

14.33.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	Basic Forms, Forms
Kontext	Unmittelbar in `<body>` sowie in `<applet>`, `<blockquote>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<iframe>`, `<ins>`, `<li>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`
Erlaubte Inhalte	Blockelemente, Inlineelemente (außer `<form>` selbst), in `transitional` auch Inlineelemente
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01, XHTML 1.0, 1.1; XHTML Basic **abgeschafft** in XTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<form> ... </form>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
form    { display: block; }
```

14.33.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	`onreset`, `onsubmit`
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`accept`, `accept-charset`, **`action`**, `enctype`, `method`, `name`, `target`
Internet Explorer	`autocomplete` *Access-, Edit-Attribute*
DoCoMo iMode	`utn`

14.33.3 Offizielle Attribute – W3C

accept

Wert	*cdata*
Typ	Ein Media-Typ-Bezeichner
Verwendung	Optional

Das `accept`-Attribut enthält eine kommaseparierte Liste all derjenigen Media-Typen (*content types* gemäß IANA-Registrierung), die der Server in Zusammenhang mit einem File-Upload akzeptieren soll. Um beispielsweise Bilder (ohne nähere Einschränkung) oder MPEG-Videos zuzulassen, könnte folgender Wert übergeben werden:

```
accept="image/*, video/mpeg"
```

Anmerkung: Das `accept`-Attribut wird nicht zuverlässig unterstützt. Muss der Dateityp aus Sicherheitsgründen geprüft werden, so sollte dies auch durch eine hierfür geeignete Methode auf dem Server geschehen.

accept-charset

Wert	*cdata*
Typ	Encoding-Bezeichner
Verwendung	Optional

Das `accept-charset`-Attribut enthält eine durch Leerzeichen oder Komma getrennte Liste erlaubter Zeichensatz-Kodierungen in denen die Formulardaten auftreten dürfen. Dies kann beispielsweise folgende Form haben:

```
accept-charset="ISO-8859-1, ISO-8859-2, ISO-8859-3"
```

Der Defaultwert in Abwesenheit des Attributs ist der reservierte Begriff `"UNKNOWN"` – der Server geht im Normalfall von »unbekanntem« Zeichensatz aus. Das Verhalten in Falle von Daten, die nicht den übergebenen Kodierungen entsprechen, ist nicht näher definiert.

action

Wert	*URL*
Typ	ein URI-String
Verwendung	*Obligatorisch*

Das `action`-Attribut gibt den URI an, an den die Formulardaten versendet werden sollen. Wird ein leerer URI-String angegeben (oder das `action`-Attribut ganz weggelassen, was gemäß der Spezifikation jedoch nicht gestattet ist), erfolgt der Datenversand an den Basis-URI der Seite, die das Formular enthält.

Häufig wird direkt eine Anwendung adressiert, die die Daten verarbeiten soll. Ist dies beispielsweise ein Perl-Script, das im CGI-Verzeichnis des adressierten Servers lagert, so kann der URI wie folgt aussehen:

```
action="http://www.example.com/cgi-bin/formular.pl"
```

Die zu wählende Übertragungsmethode `GET` oder `POST` ist davon abhängig, ob die Zielanwendung die Daten dem Querystring oder dem Standard-Input des Servers entnehmen soll. Ist ein Versand der Formulardaten per E-Mail gewünscht, so kann auch das Mailprotokoll und die Zieladresse genannt werden. Hierfür muss zwingend `method="post"` gesetzt sein:

```
action="mailto:formulardaten@example.com"
```

enctype

Wert	*cdata*
Typ	Ein MIME-Typ-Bezeichner
Verwendung	Optional

Das `enctype`-Attribut nennt den MIME-Typ nach RFC 2045[25], mit dem die Formulardaten übertragen werden sollen.

Der Defaultwert (ohne `enctype`-Attribut) ist `enctype="application/x-www-form-urlencoded"`, bei dem übertragene Textdaten nach RFC1738[26] encodiert werden. Dieser MIME-Typ lässt jedoch keine Übertragung von Dateien über das Formular zu. Hierfür muss `enctype="multipart/form-data"` verwendet werden.

25 RFC2045: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, N. Freed/N. Borenstein, Nov. 1996 (siehe http://www.ietf.org/rfc/ rfc2045.txt).

26 RFC 1738: *Uniform Resource Locators*, T. Berners-Lee, L. Masinter, M. McCahill, Dez. 1994 (siehe http://www.ietf.org/rfc/rfc1738.txt).

Weiterhin gebräuchlich, wenn auch nicht offizieller Standard, ist `enctype="text/plain"`. Dieser Wert kann verwendet werden, sofern nur textbasierte Daten übertragen werden sollen, die nicht codiert werden brauchen (beispielsweise für Mailversand).

method

Wert	*"get"* | *"post"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `method`-Attribut bestimmt die Methode, über die die Formulardaten dem Server zur Verfügung gestellt werden. Als Werte sind hier `method="get"` und `method="post"` zulässig. Der Defaultwert ist `"get"`.

- `method="get"`
  Die `GET`-Methode verschickt die Formulardaten als Teil eines Querystrings an den URL, der im `action`-Attribut genannt wird. Da mittels `GET` nur Daten in Form von ASCII-Zeichen versandt werden können, ist gegebenenfalls eine Encodierung erforderlich.
- `method="post"`
  Die `POST`-Methode verpackt die Daten im HTTP-Header und stellt sie dem Server über Standard-Input zur Verfügung.

[»] *Hinweis:* Soll ein Dateiupload über das Formular möglich sein, so *muss* `method` auf `"post"` gesetzt sein.

name

Wert	*nmtoken*
Typ	Ein String in Form eines Nametokens
Verwendung	Optional
Gültigkeit	**eingeführt** in HTML 4.01 **abgeschafft** in XHTML 1.0, 1.1

Mit dem `name`-Attribut kann ein Bezeichner für das Formular vergeben werden, der dieses so wie ein `id`-Bezeichner für Scripte adressierbar macht.

Im Gegensatz zum `name`-Attribut bei Inputelementen steht das `name`-Attribut des Formulars nicht unmittelbar mit dem Datenversand in Verbindung und wurde daher in XHTML zugunsten des `id`-Attributs abgeschafft.

target

Wert	*"_blank"* | *"_parent"* | *"_self"* | *"_top"* | cdata
Typ	Festgelegte Bezeichner oder Fensternamen
Verwendung	Optional
Gültigkeit	seit HTML 4.0 (nicht in XHTML 1.0 strict)

Mittels des `target`-Attributs wird bestimmt, in welchem Fenster oder Frame die Serverantwort auf die verschickten Formulardaten ausgegeben werden sollen. Hierfür können die Standard-Framebezeichner oder ein Bezeichner eines vorhandenen Frames oder Browserfensters übergeben werden.

- `_self` (Default)
  Die Serverantwort erfolgt im gleichen Fensterobjekt, in der sich auch das Formular befindet.
- `_parent`
  Die Serverantwort erfolgt im unmittelbar dem aktuellen Frame übergeordneten Frameobjekt des aktuellen Framekontexts.
- `_top`
  Die Serverantwort erfolgt im hierarchisch obersten Frame des aktuellen Framekontexts (auch als »Hauptfenster« bezeichnet; der Frameset wird entladen).
- `_blank`
  Die Serverantwort erfolgt in einem sich außerhalb des aktuellen Framekontexts neu öffnenden Fenster.
- *nametoken*
  Ein frei gewählter Bezeichner führt die Serverantwort im gleichnamigen Fensterobjekt aus (falls vorhanden – dies kann ein benanntes Frameobjekt oder ein frei schwebendes Fenster sein) oder öffnet ein Browserfenster, das anschließend unter diesem Bezeichner ansprechbar ist.

Hinweis: Der vordefinierte Bezeichner `"_blank"` öffnet ebenfalls ein neues Fenster, vergibt dabei aber keinen Bezeichner, so dass dieses Fenster in Folge über Links *nicht* erneut ansprechbar ist. **[«]**

Besonderheit in XHTML

Die Verwendung des `target`-Attributs erfordert die Einbindung des XHTML-Moduls »Target«. In XHTML 1.0 »*strict*« ist das Attribut nicht enthalten.

14.33.4 Proprietäre Attribute – Internet Explorer

autocomplete

Wert	*"on"* | *"off"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	seit IE5

Das `autocomplete`-Attribut erlaubt bzw. unterbindet das Autocomplete-Verhalten für Formulare im Internet Explorer. Wird `autocomplete="on"` gesetzt, darf der Browser im Nutzerprofil gespeicherte Daten zum Füllen der Formularfelder heranziehen. Der Wert `autocomplete="off"` zwingt den Nutzer, die Eingaben in jedem Fall selbst vorzunehmen.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.33.5 Proprietäre Attribute – DoCoMo iMode

utn

Wert	*"utn"*
Typ	Festgelegter Werte
Verwendung	Optional
Gültigkeit	iMode 3,0, 4.0; iMode XHTML 1.0, 1.1

Identifizierung: Durch Setzen des `utn`-Attributs wird vom Server eine persönliche Teilnehmeridentifizierung abgefragt, also beispielsweise das Versenden der Seriennummer des Endgeräts bzw. zusätzlich seiner FOMA-Card[27] (iMode). Das Attribut wird nicht von allen Endgeräten unterstützt. Der Teilnehmer kann das Versenden der Kennung verweigern.

27 FOMA ist ein DoCoMo-Akronym für »Freedom of Mobile Multimedia Access«.

14.33.6 Erläuterung zum Element

Der <form>-Container gruppiert als Blockelement Formularelemente zu einem Webformular und bestimmt das Ziel des Datenversands, den Übertragungsmodus und den MIME-Typ der verschickten Daten.

Es dürfen mehrere, voneinander unabhängige Formulare in der gleichen HTML-Seite koexistieren, die für Scripte durch ID leichter adressierbar gemacht werden können (alle Formulare sind aber stets auch über document.forms zugänglich). Die zu einem Formular gehörenden Eingabeelemente müssen sich innerhalb des jeweiligen <form>-Containers befinden.

```
<form>
   <!-- alle Formularelemente dieses Formulars hier -->
</form>
```

Attribute und Datenversand

Das Formular besitzt eine Reihe von Attributen, von denen allein das action-Attribut Pflicht ist – es dient dazu, das Ziel des Datenversandes zu bezeichnen. Je nach Art der versendeten Daten mag auch das enctype-Attribut von Bedeutung sein: Sobald mittels <input type="file"/> ein Dateiupload erfolgen soll, muss der Wert enctype="multipart/form-data" sowie method="post" gesetzt werden.

Die HTTP-Methode des Datenversands ist defaultmäßig auf GET eingestellt. Wird die sicherere Variante POST vorgezogen, so muss method="post" angeordnet werden. Bei einer Übermittlung mit der GET-Methode werden die Formulardaten an den URL des CGI-Scripts durch ein Fragezeichen getrennt angehängt, die Name-Wertpärchen durch Ampersand verbunden.

Dieser so genannte »Querystring« besitzt also die Struktur

```
?Feldname_1=Feldwert_1&Feldname_2=Feldwert_2
```

Leerzeichen innerhalb der Werte werden durch + ersetzt, nicht-alphanumerische ASCII-Werte durch %-Zeichen, gefolgt von der ASCII-Codezahl, dargestellt. Auf dem Server stehen die Formulardaten in der Umgebungsvariable QUERY_STRING zur Verfügung.

Bei der "post"-Methode werden die Daten nicht an den URL angehängt, sondern in einem eigenen Datenpaket an das gewählte CGI-Script übermittelt, das diese über den »Standard-Input« einliest. Bei der Wahl der geeigneten Übertragungsmethode müssen deren Vor- und Nachteile abgewogen werden.

- GET ist schneller als POST, da die Übermittlung in einem Schritt stattfindet.

- CGI-Scripts, die Daten direkt aus dem QUERY_STRING empfangen, sind (ein wenig) einfacher zu programmieren – das spricht für die GET-Methode.
- Da die Formulardaten bei der GET-Methode im URL enthalten sind, lässt sich der URL mitsamt den Formulardaten in den Bookmarks ablegen – praktisch, um beispielsweise eine vorformulierte Suchmaschinenquery aufzubewahren. Für den Site-Betreiber ergibt sich die Möglichkeit, Links mit festgelegten Formulardaten auf einer Webseite zu plazieren.
- Die Länge des QUERY_STRING ist bei vielen Serversystemen beschränkt. Dies kann dazu führen, dass bei der Übertragung mittels GET Daten verlorengehen oder abgeschnitten werden. Für zu erwartende größere Datenmengen ist POST daher die bessere Methode. Für Dateiupload ist POST obligatorisch.
- Die angeforderten URLs werden in den Logfiles der Server und Proxy-Server festgehalten. Hierdurch können Daten, die mit GET übertragen werden, leicht unterwegs abgefangen oder eingesehen werden. Ebenso denkbar sind Modifikationen des Querystrings, die für Angriffe auf den Server eingesetzt werden können.

[»] *Hinweis:* Sensible Daten wie Adressen oder Kreditkarteninformationen sollten stets mit POST übertragen werden. Auch damit ist zwar keine totale Sicherheit gegeben, das Ausspionieren erfordert aber geringfügig mehr Aufwand.

Inhaltsmodell

Das Inhaltsmodell des <form>-Container variiert je nach DTD-Variante stark. Allgemein gilt jedoch: Formulare dürfen nicht verschachtelt werden, da dies die Zuordnung der Eingabefelder verhindern würde (in der HTML-DTD ist dies per Elementausschluss geregelt).

- Die »transitional«-DTDs von HTML und XHTML erlauben Text (#PCDATA) und beliebige Inline- und Blockcontainer (bis hin zum Inlineframe) unmittelbar in <form>.
- Die »strict«-DTDs gestatten lediglich den <script>-Container sowie die für die DTD geltenden Blockcontainer. Es darf gemäß der »strict«-DTD also weder Text noch Inlinecontent unmittelbar im Formularelement erscheinen.

Folgendes Beispiel wäre daher in HTML 4.01 »strict« nicht valide (ebenso wenig in XHTML 1.0 strict), in einer »transitional«-DTD dagegen völlig legal:

```
<form action="beispiel.cgi">
   Ihr Name: <input type="text">
</form>
```

Grundsätzlich ist es empfehlenswert, sich eher nach den *strict*-Gepflogenheiten zu richten. Bereits durch Einfügen eines <p>-Containers gilt das Beispiel in allen DTDs:

```
<form action="beispiel.cgi">
   <p>Ihr Name: <input type="text"></p>
</form>
```

Alternativ können auch <fieldset>-Container oder -Container verwendet werden. Erstere stellen eine logische Gruppierung von Formularelementen dar und besitzen eine Defaultpräsentation (Rahmen um die Inhalte), was jedoch vielleicht nicht erwünscht ist. Auch Tabellen sind gestattet, wenn hiervon auch im Sinne der Barrierefreiheit eher abgeraten werden muss.

Die Präsentation des <form>-Containers ist abgesehen von seiner Blockeigenschaft neutral. Es kann jedoch mittels CSS-Anweisungen eine beliebige Darstellung zugewiesen werden.

Barrierefreiheit

Formulare werden besser barrierefrei zugänglich, wenn die Zuordnung zwischen dem Namen des Formularfeldes und dem Feld selbst eindeutig ist. Dies wird durch eine Positionierung des Feldnamens direkt vor dem dazugehörigen Formularfeld erzielt. Die Verbindung zwischen Beschriftung und Formularfeld darf nicht unterbrochen werden (siehe WAI 10.2, BITV 10.2).

Die herkömmliche Methode, Formulare zu layouten, greift auf Tabellen zurück, was folglich als wenig barrierefrei gelten muss (Eingabefeld und Beschriftung liegen in verschiedenen Tabellenzellen, allerdings optisch benachbart).

```
<form method="get" action="#" name="formular">
<table cellpadding="3" cellspacing="0" border="1">
  <tr>
   <td align="right" width="50%">
   <b>Vorname: </b>
   </td>
   <td width="50%">
   <input name="Vorname" type="text">
   </td>
  </tr>
  <tr>
    <td align="right"><b>Nachname: </b></td>
    <td><input name="Nachname" type="text"></td>
  </tr>
        ...
        <!-- Straße, PLZ, Ort: -->
```

```
        ...
  <tr>
  <tr align="center">
    <td>
    <input value="Senden" name="Submit" type="submit">
    </td>
    <td>
    <input value="L&ouml;schen" name="Reset" type="reset">
    </td>
  </tr>
</table>
</form>
```

Das Ergebnis ist optisch brauchbar, allerdings nur in visuell orientierten Webbrowsern uneingeschränkt einsetzbar (siehe Abbildung 14.38; den hier sichtbaren Tabellenrahmen würde man sicherlich deaktivieren).

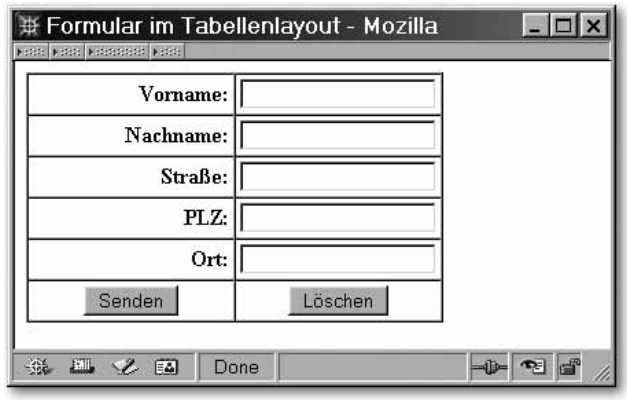

Abbildung 14.38 Ein Formular im Tabellenlayout

Eine potentielle Barriere ist ein solches Layout für manche Mobilgeräte und für Screenreader (dass Letztere mit Tabellen jedoch gar nicht zurecht kämen, gehört in das Reich der Legenden). Günstiger und auch codesparender ist das Layout per `<div>`, wobei gleich mittels `<label>`-Containern ein Bezug zwischen einem Input und seiner Beschriftung geschaffen werden kann.

Das gleiche Formular könnte auf folgende Weise barrierefreier aufgebaut werden, wobei die `<div>`-Container die Rolle der Tabellenzeilen übernehmen. Die Breitenzuweisung kann per CSS vorgenommen werden, wobei das `<form>`-Element die gewünschte Gesamtbreite erhält (dies hätte in der Tabellenversion der `<table>`-Container übernommen) und die Inputs selbst eine fixe Breite.

```
<form method="get" action="#" name="formular">
<div>
   <label for="vorname">Vorname: </label>
   <input id="vorname" name="Vorname" type="text">
</div>
<div>
   <label for="nachname">Nachname: </label>
   <input id="nachname" name="Nachname" type="text">
</div>
    ...
<div style="text-align:center">
   <input value="Senden" name="Submit" type="submit">
   <input value="L&ouml;schen" name="Reset" type="reset">
<div>
</form>
```

Die eigentlich wünschenswerte Breitenzuweisung an das `<label>`-Element per CSS schlägt in Mozilla und Opera fehl (es müsste als Blockelement umdefiniert werden[28] – nur Internet Explorer unterstützt eine Breitenangabe auch ohne dies). Der vom W3C vorgesehene Mittelweg, das Label als Inline-Block zu definieren (mit `display:inline-block`), scheitert ebenfalls an der Mozilla-Unterstützung. Mozilla bietet immerhin das Property `display:-moz-inline-box`, was `display:inline-block` entspricht. Die Rechtsausrichtung des Labeltextes muss jedoch vom Formular ausgehend vorgenommen werden.

Immerhin lässt sich mit folgendem CSS browserübergreifend eine akzeptable Optik erreichen (siehe Abbildung 14.39):

```
form       { width:300px;
             background-color:#eee;
             border:1px dotted #aaa;
             padding:5px;
             text-align:right; }

form div   { padding:3px; }

label      { font-weight:bold; }

input      { width:130px;
             margin-left:5px; }
```

28 Ein interessanter Weg, dennoch zum Ziel zu kommen, ist im Artikel *Prettier Accessible Forms* von Nick Rigby beschrieben: http://www.alistapart.com/articles/prettyaccessibleforms/.

Abbildung 14.39 Formular in -Layout

[»] *Hinweis:* Anstelle von `<div>` können auch Listenelemente `<li>` innerhalb von `<ol>`-Containern eingesetzt werden. Eine weitere gangbare Methode des Layoutens von Formularen mit Hilfe von Definitionslisten `<dl>` ist auf Seite 1 in Zusammenhang mit dem Element `<dt>` beschrieben.

14.34 frame

Das `<frame>`-Element bestimmt Quelldokument und Eigenschaften eines innerhalb eines Framesets definierten Fensterbereiches (»Frame«, »Rahmen«).

14.34.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Leeres Element
XHTML-Modul	Frames
Kontext	Innerhalb eines `<frameset>`-Containers
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 4.0 / 4.01 (fs), XHTML 1.0 (fs) **nicht enthalten** in HTML 4.0/4.01 (tr/strict) XHTML 1.0 (tr/strict), 1.1, 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

`<frame` [Attribute] `>` (HTML-Kontext)

`<frame` [Attribute] `/>` (XHTML-Kontext)

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

–

14.34.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `style`, `title`
XHTML	`class`, `id`, `style`*, `title` *`style`-Attribut nicht im XHTML-Modul »Frames«

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforeunload`, `onblur`, `onfocus`, `onload`, `onresize`, `onunload`

Elementspezifische Attribute

W3C	`frameborder`, `longdesc`, `marginheight`, `marginwidth`, `name`, `noresize`, `scrolling`, `src`
Internet Explorer	`align`, `allowtransparency`, `application`, `bordercolor`, `height`, `width` *Access-, Databinding-, Edit-Attribute*
Netscape	`bordercolor`

14.34.3 Offizielle Attribute – W3C

frameborder

Wert	*"1" | "0"*
Typ	Festgelegte Werte
Verwendung	Optional

Das `frameborder`-Attribut legt fest, ob ein Rahmen (Border) um den durch das `<frame>`-Element bezeichneten Fensterbereich gelegt werden soll oder nicht. Der Wert `frameborder="0"` deaktiviert den Rahmen. Defaultwert ist `"1"`.

Anmerkung: Der Internet Explorer unterstützt zusätzlich die Werte `"yes"` und `"no"`, die aber von anderen Browsern nicht erkannt werden. Die Verwendung von `"no"` kann daher dazu führen, dass der Wert von standardkonformen Browsern ignoriert und ungewollt die Frameborder gezeigt wird (Defaultverhalten).

CSS-Alternative: keine

longdesc

Wert	*URI*
Typ	Ein URI-String
Verwendung	Optional

Barrierefreiheit: Das `longdesc`-Attribut richtet sich primär an User Agents, die keine Frames unterstützen. Es nennt ergänzend zu möglichen Informationen aus dem `title`-Attribut einen URI zur Verknüpfung mit einer Ressource, die eine ausführlichere Beschreibung des Frameinhalts enthält.

marginheight

Wert	*number*
Typ	Eine Ganzzahl als Pixelwert
Verwendung	Optional

Das `marginheight`-Attribut legt für den oberen und unteren Dokumentrand den Abstand in Pixeln zwischen Inhalten und Framerand fest. Es hat dabei Vorrang vor dem Defaultwert. Es muss ein positiver Wert größer oder gleich Null übergeben werden.

Der Defaultwert beträgt 8 Pixel.

CSS-Alternative:

```
 /* nur im Internet Explorer */
frame {padding-top: 8px; padding-bottom: 8px;}

/* Mozilla & Co. (nur pauschal für gesamten Frameset) */
frameset { margin-top: 8px; margin-bottom: 8px;}
```

Für Opera existiert keine CSS-Alternative.

marginwidth

Wert	*number*
Typ	Eine Ganzzahl, als Pixelwert interpretiert
Verwendung	Optional

Das `marginwidth`-Attribut legt für den linken und rechten Dokumentrand den Abstand in Pixeln zwischen Inhalten und Framerand fest. Es hat dabei Vorrang vor dem Defaultwert. Es muss ein positiver Wert größer oder gleich Null übergeben werden.

Der Defaultwert beträgt 8 Pixel.

CSS-Alternative:

```
/* nur im Internet Explorer */
frame {padding-left: 8px; padding-right: 8px;}

/* Mozilla & Co. (nur pauschal für gesamten Frameset) */
frameset { margin-left: 8px; margin-right: 8px;}
```

Für Opera existiert keine CSS-Alternative.

name

Wert	*nmtoken*
Typ	Ein Bezeichner in Form eines `NMTOKEN`
Verwendung	Optional

Das `name`-Attribut bestimmt den Bezeichner des Frames, der zur Frameadressierung mittels des `target`-Attributs eines Hyperlinks benötigt wird. Wird kein `name`-Attribut vergeben, so kann der Frame nicht von einem anderen Frame aus per Hyperlink angesprochen werden.

Der Attributwert ist nicht *case-sensitive* – eine Unterscheidung zwischen zwei Framebezeichnern allein durch die Schreibweise ist daher nicht möglich. Gleichlautende Bezeichner `name="OBEN"` und `name="oben"` würden somit als identisch betrachtet.

Anmerkung: Laut Spezifikation müssen selbst gewählte Framebezeichner mit einem Buchstaben beginnen. Der Beginn mit Unterstrich ist den reservierten Framebezeichnern vorbehalten.

noresize

Wert	*"noresize"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `noresize`-Attribut legt fest, dass der betroffene Frame vom User nicht in der Größe verändert werden kann. Das Attribut ist in HTML ein Marker-Attribut (ohne erforderliche Wertzuweisung), muss in XHTML aber mit Wertzuweisung (`noresize="noresize"`) geschrieben werden.

Anmerkung: Der IE4 kennt auch den nicht standardkonformen Wert `"resize"`, der die Größenveränderlichkeit ausdrücklich zulässt. Dies entspricht jedoch dem Defaultverhalten ohne gesetztes `noresize`-Attribut.

scrolling

Wert	*"auto"* | *"yes"* | *"no"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `scrolling`-Attribut legt fest, ob der Inhalt des betroffenen Frames gescrollt werden darf, und bestimmt damit gleichzeitig die Anwesenheit von Scrollbars.

- `scrolling="auto"` (Defaultwert)
  Das Scrollen wird zugelassen, die horizontalen oder vertikalen Scrollbars jedoch nur gezeigt, wenn dies durch die Größe (Breite, Höhe) des Frameinhalts jeweils erzwungen wird.
- `scrolling="yes"`
  Das Scrollen wird zugelassen und gleichzeitig (auch falls dies anhand der Größe des Frameinhalts nicht erforderlich ist) werden sowohl horizontale als auch vertikale Scrollbars gezeigt.
- `scrolling="no"`
  Das Scrollen wird untersagt und in keinem Fall Scrollbars gezeigt, auch wenn dies durch die Größe des Frameinhalts erforderlich wäre.

Anmerkung: Das Deaktivieren des Scrollings im `<frame>`-Element unterbindet in manchen Browsern auch die Möglichkeit, den Frameinhalt per Scripting zu scrollen.

src

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das src-Attribut legt das Startdokument fest, das dem Framebereich bei Aufruf des Framesetdokuments zugeordnet und durch diesen geladen wird. Ohne gesetztes src-Attribut (das ist durchaus legal) bleibt der Fensterbereich leer.

Anmerkung: Wenn Sie auf die Zuweisung eines Startdokuments verzichten, sollten Sie dem Frame einen name-Bezeichner geben, um ein Dokument später per Link laden zu können. Beachten Sie aber, dass ein leer bleibender Framebereich vom User höchstwahrscheinlich als Fehler betrachtet wird.

14.34.4 Proprietäre Attribute – Internet Explorer

align

Wert	*"left" | "center" | "right"| "top" | "bottom"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Gültigkeit	IE3

(Historisch) Das align-Attribut bestimmt im Internet Explorer 3.0 die Ausrichtung des Framebereichs, sofern dieser kleiner als die zur Verfügung stehende Fläche ist. Es entspricht von der Wirkung her dem align-Attribut des <table>-Containers.

allowtransparency

Wert	*"true" (oder Attribut nicht gesetzt)*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Gültigkeit	ab IE5.5

Das allowtransparency-Attribut erlaubt das Durchscheinen einer für den Framebereich per CSS definierten Hintergrundfarbe. Dies ist nur dann möglich, wenn für den <body> der im Frame gezeigten Seite die CSS-Anweisung background:transparent festgelegt wurde. Ansonsten hat die Hintergrundfarbe der Seite Vorrang (für den Frameset kann keine Hintergrundfarbe gesetzt werden).

- allowtransparency="true" (oder beliebiger anderer Wert)
  Die für den Frame definierte Hintergrundfarbe wird im gesamten Framebereich gezeigt, wenn das enthaltene Dokument selbst transparent ist. Ansonsten wird die Hintergrundfarbe nur für einen eventuellen Paddingbereich des Frames gezeigt.

Anmerkung: Bei Microsoft ist der Wert *true* spezifiziert, das Property wird jedoch für jeden beliebigen Wert (auch *false*!) auf den Zustand »true« gesetzt, so dass es lediglich auf das Setzen des Attributs an sich ankommt.

- `allowtransparency="false"` (Default)
  Vorsicht: Was *nicht* funktioniert, ist das Deaktivieren der Transparenz durch den Wert *false* (wenn auch so spezifiziert!). Setzen Sie das Attribut einfach nicht. Ohne Attribut bleibt der Paddingbereich des Frames weiß, auch wenn für ihn per CSS eine Hintergrundfarbe angeordnet ist.

[zB] *Beispiele:*

a) Der Paddingbereich wird *gelb* gezeigt. Die für den Frame definierte Hintergrundfarbe scheint ebenfalls durch das enthaltene Dokument durch:

```
<frame allowtransparency="true" src="transparent.html"
 style="padding:40px;background-color:yellow;" />
```

Für *transparent.html* gilt:

```
<body style="background:transparent"> ... </body>
```

b) Der Paddingbereich wird *gelb* gezeigt, ansonsten gilt die für das enthaltene Dokument definierte Hintergrundfarbe rot:

```
<frame allowtransparency="true" src="rot.html"
 style="padding:40px;background-color:yellow;" />
```

Für *rot.html* gilt:

```
<body style="background:red"> ... </body>
```

c) Normalzustand. Der Paddingbereich wird *weiß* gezeigt, *obwohl* für den Frame eine Hintergrundfarbe definiert ist. Auch für das enthaltene Dokument wird mit weißem Hintergrund dargestellt (da es transparent ist, könnte hier theoretisch die Framehintergrundfarbe durchscheinen):

```
<!-- Attribut allowtransparency nicht gesetzt: -->
<frame src="transparent.html"
 style="padding:40px;background-color:yellow;" />
```

Für *transparent.html* gilt:

```
<body style="background:transparent"> ... </body>
```

application

Wert	*"yes"* | *"no"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `application`-Attribut wird im Internet Explorer in Zusammenhang mit so genannten **HTML Applications**[29] (HTA) verwendet, für die reguläre Webseiten eingesetzt werden, die bei Kennzeichnung mit Endung `.hta` durch den Internet Explorer (ab Version 5.0) gesondert ausgewertet werden. Da eine HTA auf das System zugreifen kann (geringe Sicherheit), wird sie jedoch nicht von einem Webserver aus betrieben, sondern muss zunächst lokal gespeichert und anschließend ausgeführt werden.

Über das `application`-Attribut kann ausgesagt werden, ob der Frameinhalt als Teil der HTML-Applikation betrachtet werden soll (`"yes"`) oder nicht (`"no"`, Default). Wird der Wert `"yes"` gesetzt, so können alle Scripte des Frameinhalts mit den gleichen Privilegien auf das lokale System zugreifen wie der Rest der HTA. Auf diese Weise kann auf Scripte und Dokumente von verschiedenen Domains zugegriffen werden.

Hinweis: Das Attribut wird ignoriert, wenn das Dokument von einem Webserver ausgeliefert wird. Dies bedeutet, dass Scripte nicht domainübergreifend ausgeführt werden können. **[«]**

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional
Einführung	IE4

Das Attribut `bordercolor` bestimmt die Rahmenfarbe des `<frame>`-Elements. In Abwesenheit des Attributs gilt die Defaultfarbe. Der Framerahmen muss aktiviert sein, d.h. das `frameborder`-Attribut also nicht auf `"no"` oder `"0"` gesetzt sein bzw. das `framespacing`-Attribut des übergeordneten `<frameset>`-Containers nicht auf den Wert 0.

height

Wert	*number[%]*
Typ	Ganzzahl als Pixelwert oder mit Prozentzeichen
Verwendung	Optional
Einführung	IE4

29 Artikel *Introduction to HTML Applications*: http://msdn.microsoft.com/workshop/author/hta/overview/htaoverview.asp.

(Historisch) Das `height`-Attribut bestimmt im Internet Explorer 3.0 die Höhe des Framebereichs, sofern diese geringer als die zur Verfügung stehende Höhe ist.

width

Wert	*number[%]*
Typ	Ganzzahl als Pixelwert oder mit Prozentzeichen
Verwendung	Optional

(Historisch) Das `width`-Attribut bestimmt im Internet Explorer 3.0 die Breite des Framebereichs, sofern dieser schmaler als die zur Verfügung stehende Breite ist.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (IE 5.5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.34.5 Proprietäre Attribute – Netscape

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt die Rahmenfarbe des Frames. Es entspricht dem gleichnamigen Attribut des Internet Explorers.

14.34.6 Erläuterung zum Element

Das `<frame>`-Element definiert im Rahmen eines Framesets einen Fensterbereich, der als **Viewport** für ein eigenes HTML-Dokument dient. Das Element ist innerhalb von Frameset-Dokumenten erlaubt, die der *frameset*-DTD von HTML 4.01 oder XHTML 1.0 entsprechen. In der (einer *strict*-DTD entsprechenden)

Spezifikation von XHTML 1.1 ist das `<frame>`-Element nicht mehr enthalten. Ab XHTML 2.0 soll die Funktionalität von Frames durch die externe XFrames-Spezifikation übernommen werden.

Das `<frame>`-Element ist leer, darf in HTML daher also keine Endmarke erhalten. In XHTML-Dokumenten muss das Element intern geschlossen werden.

Die Aufgabe des Elements besteht in der Zuweisung von Eigenschaften an den Fensterbereich, den es definiert. Dies ist in erster Linie ein Startdokument, das beim Laden des Framesets im Fenster dargestellt wird, und in zweiter Linie ein Target-Bezeichner (Framename), der es ermöglicht, das angezeigte Dokument über einen Link mit entsprechendem Target zu wechseln.

Des Weiteren können Eigenschaften des Fensterbereichs wie die An- oder Abwesenheit eines Rahmens (*frameborder*), das Auftreten von Scrollbalken für das Inhaltsdokument (*scrolling*) oder die Größenveränderbarkeit des Bereiches (*noresize*) definiert werden.

Anwendung von CSS

Das Stylen von Frames mit CSS ist, gelinde gesagt, problematisch. Zwar ist es möglich, Frames und Framesets über Dokumentstyles, externe Styles und das `style`-Attribut beliebig CSS-Präsentationen zuzuweisen, jedoch ist die Unterstützung durch Browser höchst uneinheitlich.

Beispielsweise ist es nicht möglich, per CSS die **Frameborder** zu unterdrücken (z. B. durch `border:0`). Hierfür müssen Sie auf HTML und das `frameborder`-Attribut zurückgreifen.

Auch das CSS-Boxmodell ist für Frames nicht vollständig implementiert. Der Internet Explorer ist zwar in der Lage, eine CSS-Border beliebiger Farbe um Framebereiche und den Frameset zu ziehen, unterstützt hierbei jedoch (zumindest für das `<frameset>`-Element) nicht alle Rahmenstile (ohnehin ist die CSS-Border nicht gleichzusetzen mit der Frameborder, was in Abbildung 14.40 rechts zu sehen ist, wo die Frameborder *nicht* deaktiviert ist!). In Opera und Mozilla bleiben CSS-Borderanweisungen auf Frames oder Framesets ohne Wirkung.

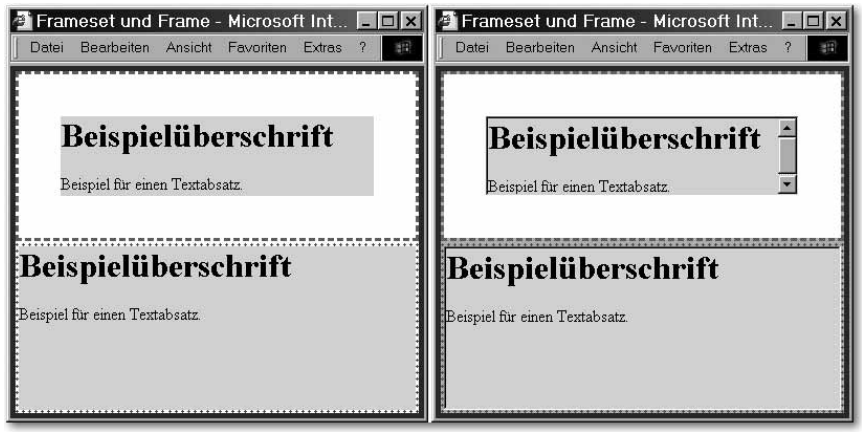

Abbildung 14.40 Frames und CSS in IE 6; links mit frameborder="0"

Die Attribute `marginheight` und `marginwidth` können im Internet Explorer durch individuelle Padding-Anweisungen für die Framebereiche selbst ersetzt werden. Die Zuweisung einer Hintergrundfarbe an den durch Padding freigehaltenen Bereich im Frame ist im Internet Explorer ab 5.5 möglich, wenn für den entsprechenden Frame das `allowtransparency`-Attribut auf `"true"` gesetzt ist.

Mozilla (und verwandte Browser) unterstützt Padding für das `<frame>`-Element nicht, ermöglicht aber als »Ausgleich« das Setzen eines globalen Margins für das gesamte Frameset (was Internet Explorer wiederum ignoriert). Hier kann dann auch (wieder nur in Mozilla) eine Hintergrundfarbe vergeben werden – das `<frameset>`-Element wird in diesen Browsern also analog zum `<body>`-Element behandelt. In Opera-Browsern haben alle diese Anweisungen keine Wirkung.

Abbildung 14.41 Frames und CSS in Mozilla; links mit frameborder="0". Beachten Sie die Hintergrundfarbe des Framesets!

Allen vorstehend gezeigten Screenshots liegt folgender Quellcode zugrunde, wobei die CSS-Anweisungen über das `style`-Attribut den betreffenden Elementen als Inlinestyles zugewiesen wurden. Das `frameborder`-Attribut wurde explizit gesetzt und abwechselnd mit den Werten `"0"` und `"1"` belegt.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Frameset und Frame</title>
</head>

<frameset rows="*,*"
 style="border:5px dashed blue; margin:20px;
 background-color:yellow;">

   <frame frameborder="1" src="15_leer.html" name="oben"
    style="border:3px dashed red; padding:40px;
    background-color:yellow;" />

   <frame frameborder="1" src="15_leer.html" name="unten"
    style="border:3px dotted green;" />

</frameset>
```

Listing 14.8 Beispieldokument mit zwei horizontalen Frames

Probleme mit Frames und Barrierefreiheit

Aus Gründen der Barrierefreiheit sind Frames eher zu vermeiden, da sie nicht auf allen Ausgabegeräten darstellbar sind. Hinzu kommen für nicht-visuelle User Agents mögliche Probleme mit der Navigation innerhalb framegestützter Websites – da zu dieser Gruppe auch Suchmaschinen zählen, ist das Indexieren solcher Sites ebenfalls erschwert.

Auch normalen Webbrowsern setzt ein Frameset Barrieren entgegen, indem nicht jeder Zustand des Framesets bookmarkfähig ist und die Inhaltsseiten nicht ohne Weiteres gedruckt werden können.

Entscheidet man sich jedoch für den Einsatz von Frames, so muss jeder Frame eindeutig mit der Angabe des zugewiesenen Bildschirmbereichs und seiner Funktion versehen werden (siehe WAI 12.2 bzw. BITV 12.2). Über das `title`-Attribut im `<frame>`-Element sollte der Framebereich darüber hinaus einen schlüssigen Bezeichner erhalten, beispielsweise »Logobereich«, »Navigation« etc. (siehe WAI

12.1 bzw. BITV 12.1). Zusätzlich noch eine inhaltliche Beschreibung mittels des `longdesc`-Attribut anzubieten, mag zwar »zukunftsträchtig« sein, wird aber derzeit erst unzureichend unterstützt.

Alternativen zu Frames

Ob das Konzept von Frames dem Grundprinzip von HTML entgegenläuft oder nicht, ist Ansichtssache. Ihre Verwendung ist durchaus legitim, sofern sie gerechtfertigt ist – ansonsten existieren einige Alternativen, die auf CSS-Basis ein optisch und zum Teil auch funktional vergleichbares Ergebnis bringen.

[»] *Hinweis:* Eine Diskussion dieses Themas finden Sie im folgenden Abschnitt über das `<frameset>`-Element.

14.35 frameset

Das `<frameset>`-Element dient zur Definition der Aufteilung des Browserfensters in einzelne Fensterbereiche, die als Frames (Rahmen) bezeichnet werden.

14.35.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Strukturcontainer
XHTML-Modul	Frames
Kontext	Das Element tritt als Kindelement des Wurzelelements `<html>` alternativ zum `<body>`-Container auf
Erlaubte Inhalte	Beliebige Anzahl `<frameset>`-Container und `<frame>`-Elemente sowie ein optionaler `<noframes>`-Container
Offizielle W3C Spezifikationen	HTML 4.0 (fs), 4.01 (fs), XHTML 1.0 (fs) **nicht enthalten** in HTML 4.0 / 4.01 (tr/strict), XHTML 1.0 (tr/strict), 1.1, 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<frameset> ... </frameset>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

–

14.35.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `style`, `title`
XHTML	`class`, `id`, `style`*, `title` *`style`-Attribut nicht im XHTML-Modul »Frames«

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	`onload`, `onunload`
Internet Explorer Events	`onafterprint`, `onbeforeprint`, `onbeforeunload`, `onblur`, `onfocus`

Elementspezifische Attribute

W3C	`cols`, `rows`
Internet Explorer	[`border`*], `bordercolor`, `frameborder`, `framespacing` *Access-, Edit-Attribute* *eigentl. NN; wird inoffiziell unterstützt
Netscape	`border`, `bordercolor`, [`frameborder`*] *eigentl. IE; wird inoffiziell unterstützt

14.35.3 Offizielle Attribute – W3C

cols

Wert	* | *number[* | %] [, * | number[* | %]]*
Typ	Breitenangabe in Pixel, Prozent oder mit *-Platzhalter
Verwendung	Optional

Das `cols`-Attribut (*columns,* Spalten) nennt die absoluten oder relativen Breiten der vertikalen Teilung des Browserfensters/Frames bzw. das Verhältnis der Breiten zueinander. Es können durch Komma getrennt mehrere Werte übergeben werden, die die Breiten der Framebereiche von links nach rechts beschreiben. Wird nur ein Wert übergeben, so wird dieser stets als 100 % (volle Breite) interpretiert.

Zahlenwerte werden als absolute Breitenangabe in Pixel angesehen, Zahlenwerte mit folgendem Prozentzeichen als prozentuale Breitenangabe, die sich auf die

verfügbare Breite bezieht. Zusätzlich ist die Verwendung des Asterisk (*) als Platzhalter erlaubt, der für eine beliebige Breite steht. Asterisken mit vorangestelltem Multiplikator stehen für eine vertikale Teilung im entsprechenden Verhältnis.

Werden `rows`- und `cols`-Attribute kombiniert, so entspricht die Anzahl der definierten Framebereiche dem Produkt der Anzahl der Argumente beider Attribute – werden beispielsweise zwei Spalten (*cols*) und drei Zeilen (*rows*) definiert, so erzeugt dies sechs (2 x 3) Framebereiche.

[zB] *Beispiele:*

`cols="150, *"`
bewirkt eine Teilung in zwei Bereiche, von denen der linke 150 Pixel breit ist, während die Breite des rechten undefiniert ist, so dass dieser stets den Rest des Fensters annimmt.

- `cols="30%, 70%"`
  bewirkt eine Teilung in zwei Bereiche, von denen der linke 30 % und der rechte 70 % des zur Verfügung stehenden Raums einnimmt.
- `cols="*, 600, *"`
  bewirkt eine Teilung in drei Bereiche, von denen der mittlere 600 Pixel breit ist, während die Breite des linken und rechten Bereiches undefiniert ist. Auf beide entfällt jeweils die Hälfte des restlichen verfügbaren Raums. Dies bewirkt eine horizontale Zentrierung eines Frames fester Breite.
- `cols="*, 2*, 3*"`
  bewirkt eine Teilung in drei Bereiche, von denen der rechte dreimal, der mittlere doppelt so breit wie der linke ist. Die absoluten Breiten hängen von der Breite des zur Verfügung stehenden Raums ab.

[»] *Hinweis:* Werden nur Pixelwerte übergeben, erfolgt die Teilung im Verhältnis der Werte zueinander. Kombinieren Sie daher Pixelwerte stets mit relativen Werten (Platzhalter).

- `cols="200, 600"`
  bewirkt eine Teilung in zwei Bereiche im Verhältnis 1 zu 3 – die Breitenangabe entspicht der Intention daher nur dann, wenn der zur Verfügung stehende Raum zufällig 800 Pixel Breite besitzt.

rows

Wert	* | *number[* | %] [, * | number[* | %]]*
Typ	Höhenangabe in Pixel, Prozent oder mit *-Platzhalter
Verwendung	Optional

Das `rows`-Attribut (*rows*, Zeilen) nennt die absoluten oder relativen Höhen der horizontalen Teilung des Browserfensters/Frames bzw. das Verhältnis der Höhen zueinander. Es können durch Komma getrennt mehrere Werte übergeben werden, die die Breiten der Framebereiche von oben nach unten beschreiben. Wird nur ein Wert übergeben, so wird dieser stets als 100 % (volle Höhe) interpretiert.

Hinweis: Die Interpretation der Werte und ihrer möglichen Kombinationen entspricht derjenigen beim `cols`-Attribut. **[«]**

Beispiele: **[zB]**

`rows="80, *"`
bewirkt eine Teilung in zwei Bereiche, von denen der obere 80 Pixel hoch ist, während die Breite des unteren undefiniert ist, so dass dieser stets die restliche Höhe des Fensters annimmt.

- `rows="20%, 80%"`
  bewirkt eine Teilung in zwei Bereiche, von denen der obere 20 % und der untere 80 % der zur Verfügung stehenden Höhe einnimmt.
- `rows="*, 450, *"`
  bewirkt eine Teilung in drei Bereiche, von denen der mittlere 450 Pixel hoch ist, während die Höhe des oberen und unteren Bereiches undefiniert ist, so dass auf beide jeweils die Hälfte des restlichen verfügbaren Höhe entfällt. Dies bewirkt eine vertikale Zentrierung eines Frames fester Höhe.

14.35.4 Proprietäre Attribute – Internet Explorer

bordercolor

Wert	*#rrggbb | farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt im Internet Explorer die Rahmenfarbe der eingeschlossenen `<frame>`-Elemente. In Abwesenheit des Attributs gilt die Defaultfarbe. Framerahmen müssen aktiviert sein, d. h. das `frameborder`-Attribut von `<frame>` darf nicht auf `"0"` gesetzt sein bzw. die (ebenfalls proprietären)

Attribute `frameborder` und `framespacing` von `<frameset>` nicht auf den Wert `"no"` bzw. `"0"`.

Anmerkung: Es existiert ein gleichnamiges, proprietäres Attribut für Netscape. Das Attribut `bordercolor` wird somit auch in Mozilla/Firefox unterstützt.

frameborder

Wert	*"yes" | "no" | "1" | "0"*
Typ	Festgelegte Werte
Verwendung	Optional

Das `frameborder`-Attribut legt im Internet Explorer fest, ob ein Rahmen (3D-Border vom Typ `"inset"`) um die im `<frameset>` enthaltenen Fensterbereiche gelegt werden soll. Der Wert `frameborder="0"` deaktiviert die 3D-Rahmen der enthaltenen Framebereiche (auch ihrer untergeordneten Framesets), nullt jedoch nicht den Framespacing-Abstand. Defaultwert ist `"1"`.

[»] *Hinweis:* Das Attribut wird ebenfalls von Mozilla/Firefox unterstützt, *nicht* jedoch von Opera!

Anmerkung: Es existiert ein offizielles namensgleiches Attribut `frameborder` für die dem Frameset untergeordneten `<frame>`-Elemente, das die Frameborder ausgehend von diesen deaktivieren kann. Verwenden Sie dieses, wenn Sie standardkonformes HTML oder XHTML erzeugen wollen.

framespacing

Wert	*number*
Typ	Eine Ganzzahl, als Pixelwert interpretiert
Verwendung	Optional

Das `framespacing`-Attribut steuert im Internet Explorer die Breite der Zwischenräume zwischen den Frames und somit indirekt die Breite der Frameborder. Es wird ein numerischer Wert übergeben, der als Pixelwert interpretiert wird. Der Defaultwert beträgt 2 Pixel.

Das Attribut wird von Opera inoffiziell unterstützt.

Anmerkung: Das `framespacing`-Attribut entspricht für sich allein in der Wirkung weitestgehend dem `border`-Attribut von Netscape/Mozilla. Da bei *aktivierter* Frameborder im Internet Explorer jedoch der Borderbreite 2 Pixel hinzugeschlagen werden, hat die Angabe `framespacing="8"` optisch die gleiche Breite wie die Angabe `border="10"` in Netscape/Mozilla.

Hinweis: Wenn Sie das Attribut `border="0"` setzen (eigentlich von Netscape stammend), brauchen Sie `framespacing` ***nicht*** zu verwenden. Gleiche Wirkung erzielen Sie mit `frameborder="0"` ***und*** `framespacing="0"`. **[«]**

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.35.5 Proprietäre Attribute – Netscape

border

Wert	*number*
Typ	Eine Ganzzahl, als Pixelwert interpretiert
Verwendung	Optional

Das Attribut `border` bestimmt in Netscape die Rahmenbreite der eingeschlossenen `<frame>`-Elemente. In Abwesenheit des Attributs gilt die Defaultbreite. Der Wert `border="0"` unterdrückt sämtliche Framerahmen und nullt den Abstand zwischen den einzelnen Framebereichen. Dieser bliebe ansonsten auch dann sichtbar, wenn für die `<frame>`-Elemente deren `frameborder`-Attribute auf `"0"` gesetzt sind.

Das Attribut `border` wird in Mozilla/Firefox derzeit weiterhin unterstützt. Da auch inzwischen Opera und Internet Explorer (seit 4.0) `border` unterstützen, können Sie es bedenkenlos einsetzen.

Achtung: Es gibt in Mozilla/Firefox ansonsten keine standardkonforme Methode, die Abstände zwischen Framebereichen auf Null zu setzen! **[!]**

Anmerkung: Das `border`-Attribut entspricht in der Wirkung weitestgehend dem Attribut `framespacing` im Internet Explorer.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt in Netscape die Rahmenfarbe der eingeschlossenen `<frame>`-Elemente. In Abwesenheit des Attributs gilt die Defaultfarbe. Framerahmen müssen *aktiviert* sein, d.h. das `frameborder`-Attribut von `<frame>` darf nicht auf `"0"` gesetzt sein bzw. das (ebenfalls proprietäre) `border`-Attribut von `<frameset>` nicht auf den Wert `"0"`.

Das Attribut wird in Mozilla/Firefox weiterhin unterstützt. Ist das gleichnamige, nicht standardkonforme Attribut für `<frame>` ebenfalls gesetzt, so hat dieses gegebenenfalls Vorrang.

Anmerkung: Internet Explorer kennt ein gleichnamiges, proprietäres Attribut.

14.35.6 Erläuterung zum Element

Frames sind heutzutage im Netz zwar nicht mehr allgegenwärtig, dennoch basieren noch immer viele Webauftritte in ihrem Layout auf Framesets.

Frames bewirken eine vollständige Aufteilung des Browserfensters in Bereiche, in denen jeweils ein selbstständiges HTML-Dokument dargestellt werden kann. In diesem Sinne ist das Browserfenster selbst tatsächlich ebenfalls ein Frame. Andersherum verhält sich jeder Frame wie ein eigenes Browserfenster. Er kann gescrollt werden, er kann auf andere HTML-Seiten verlinkt und der Inhalt unabhängig von den anderen Frames ausgetauscht werden.

Von der grundsätzlichen Organisation ist ein Framescreen einer HTML-Tabelle nicht unähnlich. Wir haben hier den Begriff Zeilen (`rows`) und Spalten (`cols`), aus denen eine Art »Gitternetz« resultiert, welches das Browserfenster aufteilt.

Für die Anweisungen, wie das Browserfenster in Frames aufzuteilen ist, wird ein eigenes Dokument benötigt: das »Frameset«-Dokument. Für einen kompletten Framescreen sind also auf jeden Fall mehrere HTML-Dokumente zu laden: das Frameset-Dokument und die in jedem Frame darzustellenden HTML-Seiten. Ein Screen mit vier Frames benötigt daher fünf Dokumente.

CSS statt Frames?

Geht es um die rein präsentative Wirkung von Frames (beispielsweise feststehende Menü- und Logobereiche bei scrollbarem Inhaltsbereich), existieren CSS-

basierte Lösungen, um dies innerhalb eines einzigen HTML-Dokuments darzustellen.[30] Einigen zu Recht bemängelten Problemen, die Framesets aufwerfen, kann hiermit abgeholfen werden:

- **Schlechte Bookmarkbarkeit**
  Frames lassen nur das Bookmarken der Framesetseite zu. Deren aktuelle, innere Framekonfiguration wird beim Bookmark nicht berücksichtigt.
- **Unzugänglichkeit für bestimmte User Agents** (auch Suchmaschinen)
  Frameinhalte sind über das `src`-Attribut und nicht über eine `href`-Beziehung eingebunden, die von Suchmaschinen o.ä. nicht als Links verfolgt werden können.
- **Ausdruck von Inhalten**
  Zum Drucken muss der zu druckende Frame im Fokus sein, was nicht immer gewährleistet ist, wenn die Browser-Druckfunktion ausgelöst wird.
- **Darstellung außerhalb des Kontextes**
  Eine Inhaltsseite außerhalb des Framekontextes ist dann nicht navigierbar, wenn die Navigation ausschließlich über den Frameset zur Verfügung gestellt wird. Mindestens eine Hilfsnavigation sollte vorhanden sein.

Für CSS-standardkonforme Browser (leider zählen alle Internet Explorer bis Version 6.0 *nicht* dazu – beim Internet Explorer 7 ist diese Methode hingegen anwendbar) sind diese Lösungen verhältnismäßig einfach und basieren darauf, die gewünschten Seitenbereiche mit `position:fixed` festzusetzen. Der ebenfalls positionierte Inhaltsbereich erhält durch die Angabe `overflow:auto` die erforderlichen Scrollbars, während dies im `<body>`-Element durch `overflow:hidden` unterbunden wird (siehe Abbildung 14.42):

```
/* Einfaches Beispiel für CSS-basierten Frameersatz */

body {
   margin:0;  padding:0;
   overflow:hidden;
   }

div#header {
   position:fixed; top:0; left:0;
   width:100%; height:60px;
   background-color:#ddd;
   }
```

30 Ausführliche Beschreibungen verschiedener CSS-Lösungen als Framealternative finden Sie bei Kai Laborenz *CSS-Praxis*, Galileo Press, S. 202 ff.

```
div#navi {
   position:fixed; top:60px; left:0;
   width:150px; height:100%;
   background-color:#eee;
   }

div#footer {
   position:fixed; bottom:0; left:0;
   width:100%; height:40px;
   background-color:#ddd;
   }

div#inhalt {
   position:fixed; top:60px; left:160px; bottom:40px;
   overflow:auto;
   }
```

Listing 14.9 CSS-basierter Pseudo-Frameset mit Header, Footer und Nav-Bereich

Abbildung 14.42 Ein mit CSS erstellter Pseudo-Frameset (Mozilla)

Für ältere Internet Explorer vor Version 7.0 müssen zusätzlich Conditional Comments oder CSS-Hacks eingesetzt werden, um zu einer vergleichbaren Ausgabe zu gelangen – dies auszuführen, führt an dieser Stelle zu weit (ich möchte hier stattdessen auf die genannte Quelle in »CSS-Praxis« von Kai Laborenz verweisen).

Vielleicht doch Frames?

Frames sollten nicht generell verteufelt werden. Im Gegenteil können sie in bestimmten Anwendungsfällen die überlegene Lösung sein und werden demzufolge auch im professionellen Bereich noch immer angewendet. Als Beispiel hierfür mag die framegestützte »Developer Library« von Microsoft gelten:

`http://msdn.microsoft.com/library/`

Folgende Gründe können für den Einsatz von Frames sprechen:

- Das Informationsangebot wird ausschließlich durch framefähige User Agents genutzt und die Suchmaschinenproblematik ist irrelevant.
- Die in Frameset dargestellten Dokumente liegen auf verschiedenen oder wechselnden Servern und können daher nicht zusammengefasst werden.
- Die Dateigröße der Einzeldokumente ist relativ groß, so dass der Datenfluss durch eine Zusammenfassung in ein gemeinsames Dokument steigen würde.
- Der Zustand der Navigation ist variabel (z. B. als Explorerbaum) und müsste abgespeichert werden können.

Die Bookmarkproblematik bei Frames ist per Scripting lösbar. Hier ist eine Lösung analog zu jener des Bookmarkproblems bei AJAX-Anwendungen denkbar, die Änderungen innerhalb des Framesets im Location-Objekt des äußeren Frameset-Dokuments widerspiegelt (und zwar in `location.hash`, um das ansonsten forcierte Neuladen des gesamten Framesets zu vermeiden).

Beispiel: Das folgende Quelltextbeispiel soll die gleichen Konfiguration erzeugen wie der in Abbildung 14.42 gezeigte CSS-Nachbau. Ein standardkonformes Nullen der Frameborder ausgehend von den Framebereichen selbst entfernt jedoch in der Tat nicht wie gewünscht die Framezwischenräume (Abbildung 14.43). Der Internet Explorer zeigt in gleicher Situation seinen 2-px-Framespacingabstand, Opera das dunkelgraue Framegitterraster (ebenfalls 2 Pixel breit). **[zB]**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Frameset und Frame</title>
</head>
<frameset rows="60, *, 40">
  <frame name="header" src="head.html" frameborder="0"/>
  <frameset cols="150, *">
    <frame name="navi" scrolling="no"
     src="navi.html" frameborder="0"/>
```

```
    <frame name="inhalt" src="start.html" frameborder="0"/>
  </frameset>
  <frame name="footer" src="footer.html" frameborder="0"/>
  <noframes>
     <body>
        <p>Dies ist f&uuml; User Agents,
        die keine Frames unterst&uuml;tzen!</p>
     <body>
  </noframes>
</frameset>
```

Listing 14.10 Framesetdokument mit verschachtelten Framensets und Noframes-Bereich

Abbildung 14.43 Zwischenräume trotz Nullung der Frameborder (Mozilla)

Abhilfe für diesen Umstand ist nur unter Umgehung der Standards zu schaffen. Hierfür kann das proprietäre Netscape-Attribut `border` verwendet werden, das von Internet Explorer und Opera ebenfalls unterstützt wird:

```
...
<!-- im äußeren Frameset zusätzlich: -->
<frameset rows="60, *, 40" border="0">
...
</frameset>
```

Beachten Sie, dass das Dokument durch Setzen des nicht standardkonformen Attributs *nicht* mehr validierbar ist – die Doctype-Deklaration sollte daher entfallen. Das Attribut `border` in `<frameset>` bietet somit einfachste Methode, Framegrenzen unsichtbar zu machen (`frameborder` in `<frame>` kann entfallen).

Für Mozilla/Firefox wäre auch `frameborder="0"` ausreichend (nicht für Opera!); für Internet Explorer muss in diesem Falle zusätzlich `framespacing="0"` gesetzt werden (was wiederum für Opera bereits alleine ausreichte).

14.36 h1 – h6

Die Familie der Blockcontainer `<h1>` bis `<h6>` definieren logische Dokumentbereiche als Überschriften vom Grad 1 (oberste Hierarchiestufe) bis 6 (unterste Hierarchiestufe).

14.36.1 Elementeigenschaften

Einführung	HTML Level 0
Elementtyp	Blockcontainer
XHTML-Modul	Text
Kontext	Unmittelbar innerhalb des `<body>`-Containers sowie in `<applet>`, `<blockquote>`, `< button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<map>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`
Erlaubte Inhalte	Inlineelemente und Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<h1> ... </h1>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
h1              { font-size: 2em;
                  margin: .67em 0; font-weight: bolder }
h2              { font-size: 1.5em;
                  margin: .83em 0; font-weight: bolder }
```

```
h3              { font-size: 1.17em;
                  margin: 1em 0; font-weight: bolder }

   /* Schriftgröße <h4> entspricht normaler Textgröße: */

h4              { font-size: 1em;
                  margin: 1.33em 0; font-weight: bolder }

   /* line-height bei <h5> gleicht Schriftgröße aus: */

h5              { font-size: .83em; line-height: 1.17em;
                  margin: 1.67em 0; font-weight: bolder }
h6              { font-size: .67em;
                  margin: 2.33em 0; font-weight: bolder }
```

14.36.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	align
Internet Explorer	*Access-, Edit-Attribute*

14.36.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right" (* | *"justify")*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte des Überschriftencontainers. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet den Inhalt der Überschrift linksbündig zum Containerrand aus.
- `align="center"`
  zentriert den Inhalt der Überschrift im Container.
- `align="right"`
  richtet den Inhalt der Überschrift rechtsbündig zum Containerrand aus.
- `align="justify"`
  richtet den Inhalt der Überschrift blocksatzmäßig aus (dieser Attributwert wurde in der Vergangenheit nicht von allen Browsern unterstützt).

14.36.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.36.5 Erläuterung zum Element

Mittels der `<h1>`- bis `<h6>`-Container werden Textabschnitte gekennzeichnet, die eine Überschrift darstellen. Die Elemente sind nach Wichtigkeit gestaffelt, wobei `<h1>` den höchsten und `<h6>` den geringsten Wichtigkeitsgrad besitzen. Dies wird

defaultmäßig durch unterschiedliche Darstellungsgröße umgesetzt, womit immer auch eine Fettauszeichnung der Inhalte verbunden ist.

Die Spezifikation rät dazu, die erste Überschrift eines HTML-Dokuments stets als `<h1>` auszuzeichnen und auf den folgenden Textebenen keine Überschriftenhierarchie zu überspringen (also nicht nach `<h1>` beispielsweise `<h3>` einzusetzen).

Barrierefreiheit

Strukturelle HTML-Elemente (wie `<h1>`, `<h2>` usw.) dürfen nur in der Funktion eingesetzt werden, für die sie vorgesehen sind (laut WAI 3.5, BITV 3.5), also nicht etwa nur zum Erhöhen der Schriftgröße. Ein Screenreader wird die Überschriftenhierarchie ansagen und gegebenenfalls als Navigationshilfe verwenden.

[zB] *Beispiel:*

```
<h1>Überschrift 1</h1>
<h2>&Uuml;berschrift 2</h2>
<h3>&Uuml;berschrift 3</h3>
<h4>&Uuml;berschrift 4</h4>
<h5>&Uuml;berschrift 5</h5>
<h6>&Uuml;berschrift 6</h6>
```

Abbildung 14.44 Überschriften von <h1> (oben) bis <h6>

14.37 head

Das Element `<head>` definiert den Dokumentkopf des HTML-Dokuments als logischen Bereich. Er gehört zu den obligatorischen Strukturelementen.

14.37.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Strukturcontainer
XHTML-Modul	Structure
Kontext	Als erstes Kindelement im Wurzelelement `<html>`
Erlaubte Inhalte	Obligatorisches `<title>`-Element, optionale Elemente `<meta>`, `<script>`, `<style>`, `<base>`
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<head> ... </head>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
head          { display: none; }
```

14.37.2 Attribute (Übersicht)

Standardattribute

HTML	`lang`
XHTML	`dir`, `xml:lang`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onreadystatechange`

Elementspezifische Attribute

W3C	`profile`
Internet Explorer	`title`

14.37.3 Offizielle Attribute – W3C

profile

Wert	*URL [URL_2 URL_2 ... URL_n]*
Typ	Ein bzw. mehrere leerzeichengetrennte URI-Strings
Verwendung	Optional

Das `profile`-Attribut dient dazu, einem Dokument mittels der übergebenen URI-Strings Meta-Informationen zuzuordnen, die in externen Dokumenten liegen, so genannte »Meta-Profile«. Die Spezifikation legt derzeit lediglich fest, dass der erste übergebene URI-String auszuwerten ist, der dann auch ein Verzeichnis angeben kann, in dem die Informationen liegen. Die Berücksichtigung mehrerer übergebener URI-Strings ist als »zukünftige Erweiterung« vorgesehen.

[»] *Hinweis:* Aus `<meta>`-Elementen heraus kann über deren `scheme`-Attribut Bezug auf die Regeln des Meta-Profils genommen werden.

14.37.4 Proprietäre Attribute – Internet Explorer

title

Wert	*cdata*
Typ	Eine beliebige Zeichenkette
Verwendung	Optional

Das `title`-Attribut dient (nur im Internet Explorer) dazu, dem `<head>`-Container einen Titel zuzuweisen. Das Attribut stellt dabei keinesfalls eine Alternative zum obligatorischen `<title>`-Container dar, sondern dient Scripting-Zwecken.

[!] *Achtung:* Die offizielle Spezifikation sieht für `<head>` ***kein*** `title`-Attribut vor!

14.37.5 Erläuterung zum Element

Der `<head>`-Container gehört zu den Strukturelementen, die in XHTML zu den Pflichtbestandteilen eines Dokuments gehören. In HTML darf der `<head>`-Container (nicht aber der in ihm liegende `<title>`-Container) weggelassen werden, da der HTML-Parser ihn im Bedarfsfall ergänzt. Es ist jedoch stark anzuraten, auch für HTML das Dokument nicht in dieser Form zu verkürzen.

Der `<head>` enthält stets einen `<title>`-Container für den Dokumenttitel und optional genau einen `<base>`-Container zur Festlegung des Basis-URI. Als weitere Inhalte sind optional beliebig viele `<meta>`- und `<link>`-Tags, Script- und Stylecontainer sowie (im Head-Kontext stets leere) `<object>`-Container gestattet.

Der <head>-Container dient allgemein der Meta-Information über das Dokument. Seine Inhalte werden nicht dargestellt bzw. er darf aus diesem Grunde keine darstellbaren Inhalte besitzen.

Beispiel: **[zB]**

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Beschreibung der Seite</title>
   <!-- Meta-Elemente, Scripte, Dokumentstyles etc. -->
</head>
<body>
   ...
</body>
</html>
```

Listing 14.11 Grundstruktur eines XHTML-Dokuments

14.38 hr

Das leere Element <hr> fügt eine horizontale Trennlinie (*horizontal rule*) ein.

14.38.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Block
XHTML-Modul	Presentation
Kontext	Als Kindelement im <body> oder anderen Elementen wie <applet>, <blockquote>, <button>, , <dd>, <del>, , <fieldset>, <form>, <iframe>, <ins>, , <map>, <noframes>, <noscript>, <object>, <td>, <th>
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

<hr> (in HTML)

<hr /> (in XHTML)

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
hr      { display: block;
          border: 1px inset }
```

14.38.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecut`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `noshade`, `size`, `width`
Internet Explorer	`color`, *Access-, Edit-Attribute*
WebTV	`invertborder`

14.38.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut legt die Ausrichtung des Horizontal Rules gegenüber den Rändern des umgebenden Blockelements fest. Die Defaultausrichtung ist zentriert, wofür die Breite des Horizontal Rules jedoch auf unter 100 % oder einen entsprechenden Pixelwert gesetzt werden muss.

CSS-Alternative: Ausrichtung vom umgebenden Container ausgehend.

noshade

Wert	*"noshade"*
Typ	festgelegter Bezeichner (oder leeres Marker-Attribut)
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `noshade`-Attribut deaktiviert den für ein Horizontal Rule defaultmäßig aktiven 3D-Darstellungsmodus (»Groove«) und stellt das Element als einfarbig dunkelgraue Linie dar (RGB-Wert: `#8C8E8C`).

Der 3D-Effekt wird durch eine oben liegende dunkelgraue Schattenlinie (RGB-Wert: `#8C8E8C`) und eine unten liegende hellgraue Lichtlinie (RGB-Wert: `#C6C3C6`) erzeugt. Wird die Höhe des Horizontal Rules durch das `size`-Attribut auf über zwei Pixel Höhe vergrößert, bleibt der innere Bereich weiß.

Anmerkung: Der Internet Explorer erkennt auch die Werte `"true"` und `"false"`, die die 3D-Darstellung deaktivieren (`true`) oder explizit aktivieren (`false`). Letzteres entspricht der Defaultpräsentation des Elements. Andere Browser unterstützen diese Werte nicht.

size

Wert	*number*
Typ	Ganzzahl, als Pixelwert interpretiert
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `size`-Attribut legt die Höhe (Dicke) des Horizontal Rule in Pixel fest. Die Defaulthöhe beträgt zwei Pixel, bei der noch eine 3D-Präsentation erfolgen kann. Wird die Höhe auf 1 Pixel verringert, wird das Horizontal Rule einfarbig dunkelgrau dargestellt (RGB-Wert `#8C8E8C`).

width

Wert	*number[%]*
Typ	Ganzzahl, als Pixelwert interpretiert oder Prozentangabe
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `width`-Attribut legt die horizontale Breite des Elements in Pixel oder in Prozent der Breite des umgebenden Blockelements fest, wobei es sich meist um das `<body>`-Element handeln wird. Die Defaultbreite entspricht dem zur Verfügung stehenden Raum und beträgt somit 100 %. Das Horizontal Rule ist also von sich aus stets so breit wie möglich.

14.38.4 Proprietäre Attribute – Internet Explorer

color

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `color` bestimmt die Farbe des Horizontal Rules. In Abwesenheit des Attributs gilt die Defaultpräsentation. Wird ein `color`-Attribut vergeben, wird die 3D-Präsentation automatisch deaktiviert, als ob das `noshade`-Attribut gesetzt wäre.

Anmerkung: Das Attribut wird außer im Internet Explorer auch in Mozilla/Firefox unterstützt, nicht jedoch in Opera.

CSS-Alternative: `hr {color:#rrggbb;}`

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute:

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute:

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.38.5 Proprietäre Attribute – WebTV

invertborder

Wert	–
Typ	Leeres Marker-Attribut
Verwendung	Optional

Das `invertborder`-Attribut kehrt für WebTV-Clients die 3D-Darstellung des Horizontal Rules um.

14.38.6 Erläuterung zum Element

Das *horizontal rule* `<hr>` stammt aus den Gründerzeiten von HTML und erzeugt eine horizontale Trennlinie, um einzelne Textabsätze optisch voneinander abzusetzen. Die ursprüngliche Intention war also weniger eine semantische als eine rein präsentative.

Die Umsetzung durch die verschiedenen Browser ist notorisch uneinheitlich und macht sich bereits bei `size`-Werten über 5 unangenehm bemerkbar (vergleichen Sie Abbildung 14.45 und Abbildung 14.46). Schmalere Rules sind hier gefahrloser.

Der `<hr>`-Tag zählt zu den Blockelementen, verursacht also einen Zeilenumbruch vor und nach dem Element. Er darf nicht innerhalb von `<p>`-Containern eingesetzt werden (er unterbricht diese sonst). Gestattet ist er hingegen in Tabellenzellen als Teil des Fallback-Inhalts von `<object>`, `<applet>` und `<iframe>` sowie innerhalb anderer Blockelemente wie `<div>`.

Barrierefreiheit und das title-Attribut von <hr>

Screenreader können das Auftreten eines `<hr>` zum einen als Signal eines Wechsels in der Dokumentstruktur werten. Zum anderen können sie ein *horizontal rule* auch als reines Präsentationselement betrachten und daher nicht zur Kenntnis nehmen. Der Webautor kann die Bedeutung des `<hr>`-Elements dadurch klarstellen, dass er ihm ein aussagekräftiges `title`-Attribut zuweist. Mit einem leeren `title`-Attribut (`title=""`) wird hingegen unterstrichen, dass dem Element keine inhaltliche Bedeutung zukommt und es ignoriert werden kann.

Beispiel: **[zB]**

```
<hr size="1">

<hr noshade="noshade">

<hr width="60%">
```

```
<hr size="15">
<hr noshade="noshade" size="15">
<hr color="#ff0000">
<hr width="60%" noshade="noshade" size="15" color="#ff0000">
```

Dieser Quelltext wird in Mozilla und Internet Explorer verschieden umgesetzt:

Abbildung 14.45 <hr>-Elemente (s. Beispiel) im Internet Explorer

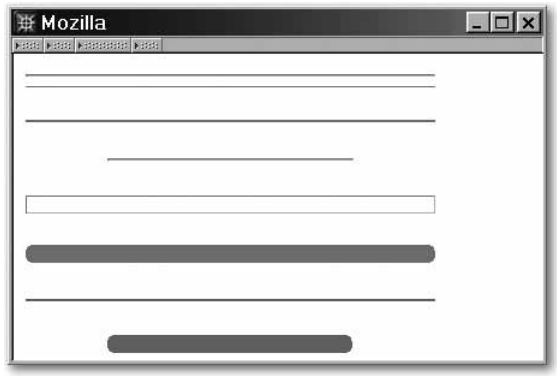

Abbildung 14.46 <hr>-Elemente im Mozilla

14.39 html

Das Element `<html>` (*hyper text markup language*) ist das obligatorische Wurzelelement eines jeden HTML- und XHTML-Dokuments. Nach alter SGML-Tradition entspricht der Elementname dem Bezeichner der DTD, die der Sprache zugrunde liegt. Der Elementname wurde auch für XHTML beibehalten.

14.39.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Strukturcontainer
XHTML-Modul	Structure
Kontext	Wurzelelement des HTML bzw. XHTML-Dokuments
Erlaubte Inhalte	Enthält die obligatorischen Strukturelemente `<head>` und `<body>`
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<html> ... </html>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

–

14.39.2 Attribute (Übersicht)

Standardattribute

HTML	`lang`
XHTML	`dir`, `id`, `xml:lang`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onreadystatechange`

Elementspezifische Attribute

W3C	`xmlns, version`
Internet Explorer	`title`

14.39.3 Offizielle Attribute – W3C

version

Wert	*cdata*
Typ	String; eine Versionsbezeichnung
Verwendung	Optional

Deprecated: Das `version`-Attribut nennt die HTML-Version des Dokuments. Es wird in HTML 4.0 zugunsten der entsprechenden Nennung der HTML-Version im Rahmen der Dokumenttyp-Deklaration abgelehnt.

```
<html version="-//IETF//DTD HTML 3.2//EN">
```

Hier wird für das Dokument die HTML Version 3.2 angezeigt. Die entsprechende Doctype-Deklaration lautet

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

Anmerkung: Das `version`-Attribut ist auch in XHTML 2.0 Teil der Spezifikation. Derzeit (Januar 2007) wird über eine normative Form des Attributwerts diskutiert, die durch entsprechende Software zuverlässig erkannt werden kann.

xmlns

Wert	*URL*
Typ	Ein URI-String, der als Namensraum dient
Verwendung	Optional
Eingeführt	XHTML 1.0

Das `xmlns`-Attribut dient für XHTML-Dokumente der Zuweisung eines XML-Namensraums an das `<html>`-Element, der von XML-Parsern beachtet wird. Der Namensraum gilt für das Wurzelelement selbst und alle von ihm abstammenden Elemente und deren Attribute, sofern ihnen nicht ein eigener Namensraum oder der Null-Namensraum zugewiesen ist. Von SGML-Parsern (d.h. im HTML-Kontext) wird das Attribut nicht beachtet.

Die gleichermaßen für XHTML 1.0 (alle DTD-Versionen), XHTML 1.1 und XHTML Basic vorgeschriebene Namensraumzuweisung lautet

```
xmlns="http://www.w3.org/1999/xhtml"
```

XHTML 1.0 und XHTML 1.1 lassen sich folglich *nicht* anhand des Namensraums unterscheiden! Hierfür kann jedoch die Doctype-Deklaration herangezogen werden.

Anmerkung: Für XHTML 2.0 wird ein anderer Namensraum verwendet. Derzeit (laut Working Draft vom 26. Juli 2006) lautet die Zuweisung

```
xmlns="http://www.w3.org/2002/06/xhtml2/"
```

14.39.4 Proprietäre Attribute – Internet Explorer

title

Wert	*cdata*
Typ	Eine beliebige CDATA-Zeichenkette
Verwendung	Optional

Das `title`-Attribut kann dem Elementcontainer einen optionalen Titel zuweisen, wird für das Element `<html>` jedoch nur durch den Internet Explorer unterstützt.

14.39.5 Erläuterung zum Element

Das `<html>`-Element ist das Wurzelelement jeden HTML- oder XHTML-Dokuments. Für XHTML-Dokumente trägt es außerdem den Defaultnamensraum, der mit Hilfe des `xmlns`-Attributs zugewiesen wird.

In HTML ist das Element als »verkürzbar« definiert und kann im Prinzip weggelassen werden (wird dann jedoch durch den Parser ergänzt). Für XHTML gilt dies selbstverständlich nicht, da das Wurzelelement für ein XML-Dokument stets Pflicht ist und Tag-Verkürzungen nicht gestattet sind.

Das `<html>`-Element besitzt stets zwei Kindelemente, nämlich `<head>` und `<body>` (in dieser Reihenfolge). Anderer Elementinhalt (beispielsweise `<script>`-Container *zwischen* `<head>` und `<body>`, eine gelegentlich anzutreffende Unsitte) oder Text (`#PCDATA`) ist nicht gestattet, Kommentare sind jedoch prinzipiell erlaubt. Dem `<html>`-Element *darf* bei einem HTML-Dokument eine Doctype-Deklaration vorangehen.

```
<html>
   <head>
      <title>Der Dokumenttitel</title>
      <!-- weiterer Headinhalt -->
   </head>
   <body>
      <!-- Bodyinhalt -->
   </body>
</html>
```

Listing 14.12 Grundstruktur HTML

Im Falle eines XHTML-Dokuments *muss* eine Doctype-Deklaration vorhanden sein und es *muss* ein Namensraum vergeben werden. Vor der Doctype-Deklaration *darf* noch die XML-Deklaration stehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE  ... >
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>Der Dokumenttitel</title>
      <!-- weiterer Headinhalt -->
   </head>
   <body>
      <!-- Bodyinhalt -->
   </body>
</html>
```

Listing 14.13 Grundstruktur XHTML

Barrierefreiheit und die Sprache des Dokuments

Die natürliche Sprache des Dokumentinhalts (*base language* bzw. *primary language*) kann bereits im Wurzelelement mittels des `lang`-Attributs signalisiert werden. In XHTML-Dokumenten geschieht dies durch das `xml:lang`-Attribut, das sich im XML-Namensraum befindet (dieser muss jedoch nicht explizit deklariert werden).

```
<html lang="de">
<!-- Natürliche Sprache des Dokuments ist Deutsch -->
</html>
```

Es wird ein Sprachbezeichner nach RFC1766 erwartet, also eine zweibuchstabige Abkürzung, die die Sprachgruppe kennzeichnet (*primary code*), gegebenenfalls erweitert durch eine ebenfalls zweibuchstabige Abkürzung, die eine länderbezogene Untergruppe bezeichnet (*subcode*):

`en-US` bezeichnet US-amerikanisches (*subcode*) Englisch (*primary code*).

Normen
Die Abkürzungen der Sprachgruppe sind in ISO639 geregelt, die der Untergruppen in ISO3166.

Auch aus Gründen der Barrierefreiheit ist das Hinzufügen des `lang`-Attributs anzuraten. Einem Screenreader gibt es Hilfestellung bei der Generierung der korrekten gesprochenen Inhalte. (Aus diesem Grunde sollten Sprachwechsel innerhalb des Dokuments ebenfalls angezeigt werden.)

Nicht zu verwechseln ist die natürliche Sprache mit dem Character-Code, wie er in der XML-Deklaration oder mit `<meta>`-Tag festgelegt werden kann. Auch die Sprachabkürzung, die im String der Doctype-Deklaration erscheint, steht nicht in Verbindung mit der Sprache des Dokuments selbst, sondern bezeichnet die Sprache, in der die DTD abgefasst ist.

Basistextrichtung des Dokuments

Ebenfalls im Wurzelelement kann die Basistextrichtung (*base text direction*) des Dokuments angegeben werden. Hierzu dient das `dir`-Attribut, das die Werte `"rtl"` (rechts nach links) und `"ltr"` (links nach rechts) annehmen kann.

Die Defaultrichtung ist »links nach rechts«. In diesem Fall braucht das Attribut nicht gesetzt werden.

```
<!DOCTYPE ... >
<html xmlns="http://www.w3.org/1999/xhtml" dir="rtl">
<head>
<title>Leserichtung rechts nach links</title>
</head>
<body>
<h1>Rechts nach Links!</h1>
<p><span>Wenn der Hund</span>
<i style="border:1px dotted #f00;">mit der Wurst</i>
<span>&uuml;bern Eckstein springt.</span><p>
<p>The title says
<span style="border:1px dotted #00f"> ,</span>
W3C in Hebrew.</p>
</body>
</html>
```

Listing 14.14 XHTML-Dokument mit umgekehrter Basistextrichtung

Das `dir`-Attribut bestimmt die Defaultausrichtung innerhalb von Blockelementen (hier: Rechts- statt Linksausrichtung). Mit Wert `"rtl"` wird gleichzeitig die Position der Interpunktion an die Leserichtung angepasst: Ein Punkt rückt daher an die linke Seite eines Satzes, unabhängig von seiner Quelltextposition (vergleichen Sie die Darstellungsposition des Ausrufezeichens in Abbildung 14.47 mit der im Quelltext).

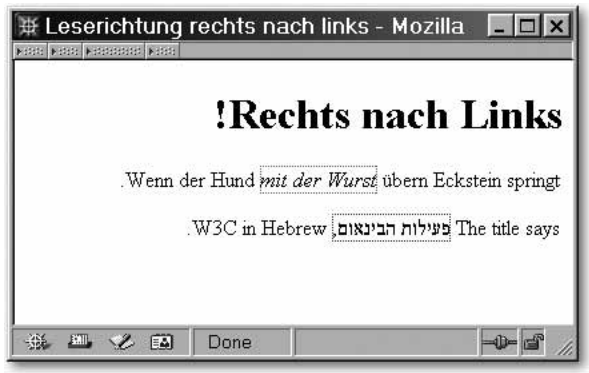

Abbildung 14.47 XHTML-Dokument mit umgekehrter Basistextrichtung

Nicht beeinflusst wird die Reihenfolge der Inlinecontainer, die Textrichtung innerhalb von Worten oder die Wortstellung innerhalb von Texten. Zu deren Regelung wird der *Bi-direktionale Algorithmus* herangezogen, der seine Information direkt aus dem Zeichentyp (Unicode-Charactercode) bezieht. Dieser Algorithmus sorgt dafür, dass beispielsweise Folgen hebräischer Zeichen von rechts nach links dargestellt werden, während Folgen »normaler« westlicher Zeichen stets in Links-nach-rechts-Ordnung verbleiben.

Erscheint ein Textabschnitt mit Bidi-Richtung in Basistextrichtung, so wird gegebenenfalls die Reihenfolge umgebender Inlinecontainer *vertauscht* (siehe der zweite Textabsatz in Abbildung 14.47 mit hebräischem Einsprengsel).

[»] *Hinweis:* Mehr über den Bi-direktionalen Algorithmus erfahren Sie in Zusammenhang mit den `<bdo>`-Element (*bidirectional override*).

DOM und Zugriff auf die Grundstruktur des HTML-Dokuments

Über das DOM ist das Wurzelelement `<html>` mittels des Properties `document.documentElement` zugänglich. Das Property enthält eine Referenz vom Typ `"node"` auf den Elementknoten. Ähnlich erhalten Sie Zugriff auf den Namensraum-URI, den Dokumenttitel und -körper.

DOM-Property	Zugriff auf
`document.documentElement`	Wurzelelement `<html>` (*node*)
`document.namespaceURI`	Namensraum-URI (*string*)
`document.doctype`	Doctype-Deklaration (*node* oder `null`)
`document.body`	Dokumentkörper `<body>` (*node*)
`document.title`	Dokumenttitel (*string*)

Tabelle 14.1 DOM-Properties zum Zugriff auf HTML-Grundstrukturen

14.40 i

Präsentationselement: Das Element <i> bewirkt die Kursivauszeichnung (*italic*) der in den Container eingeschlossenen Textinhalte.

14.40.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext:	Überall in Block- oder Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text (#PCDATA)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1; XHTML mobile profile **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	– (keine) *nicht enthalten* in HTML 4.0 mobile iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<i> ... </i>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
i     { font-style: italic }
```

14.40.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–

Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	*Access-, Edit-Attribute*

14.40.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.40.4 Erläuterung zum Element

Das Element `<i>` dient rein präsentativ der Kursivauszeichnung seines Textinhalts. Es zählt zu den so genannten »Fontstyle«-Elementen. Der `<i>`-Container ist in allen DTD-Varianten gültig, kann also auch in XHTML 1.0 »strict« und XHTML 1.1 eingesetzt werden. Der Container muss auch in HTML stets geschlossen werden.

```
<p>Ein <i>kursiv ausgezeichnetes</i> Wort ist optisch
hervorgehoben.</p>
```

```
<p>Ein <i title="echt schr&auml;g!">kursiv ausgezeichnetes</i> Wort
ist optisch hervorgehoben.</p>
```

```
<p>Ein <i>doppelt <i>kursiv gesetztes Wort</i> wird nicht</i>
kursiver.</p>
```

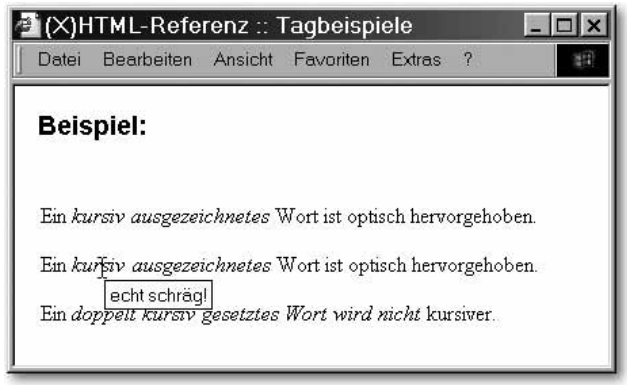

Abbildung 14.48 Kursivsetzung mit <i> im Internet Explorer (zweites Beispiel mit Title-Tooltip)

Eine dem Element entsprechende Präsentationswirkung erreichen Sie in CSS mit dem Property `font-style:italic`. Anstelle der folgenden Lösung könnte natürlich genauso gut (und codesparend) gleich `<i>` eingesetzt werden:

```
<p>Eine <span style="font-style:italic">Kursivauszeichnung </span>
ist auch mit CSS m&ouml;glich.</p>
```

Screenreader

Ob eine Kursivformatierung durch Screenreader beachtet wird, hängt von deren Einstellung (»Verbosity Scheme«) ab. Ein Screenreader ist durchaus in der Lage, über reine Textformatierungen zu informieren (»speech style announcements«), wird dies jedoch in der Grundeinstellung nicht tun.

Entspricht die Intention also einer Hervorhebung »betont«, so ist das logische Element `<em>` vorzuziehen, das ebenfalls eine Kursivauszeichnung bewirkt. In diesem Fall kann der Screenreader beispielsweise mit einer Erhöhung der Lautstärke reagieren.

Anmerkung: In XHTML 2.0 ist das Element `<i>` abgeschafft, da es sich um reine Darstellungsauszeichnung handelt. Zu diesem Zweck sollen ausschließlich Stylesheetformatierungen herangezogen werden.

14.41 iframe

Das `<iframe>`-Element (*inline frame*) definiert einen Frame in Form eines im Dokument fixierten Inlinecontainers, in dem ein externes Dokument dargestellt werden kann.

14.41.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Iframe
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Block und Inlineelemente
Offizielle W3C Spezifikationen	HTML 4.0 / 4.01 (tr/fs), XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0/4.01 strict XHTML 1.0 strict, 1.1, 2.0
Mobil-Spezifikationen	– (keine)

Syntaxbeispiel:

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

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

–

14.41.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `style`, `title`
XHTML	`class`, `id`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `onfocus`

Elementspezifische Attribute

W3C	`align`, `frameborder`, `height`, `longdesc`, `marginheight`, `marginwidth`, `name`, `scrolling`, `src`, `width`
Internet Explorer	`application`, `border`, `bordercolor`, `hspace`, `noresize`, `vspace` *Access-, Databinding-, Edit-Attribute*

14.41.3 Offizielle Attribute – W3C

align

Wert	*"top"* | *"middle"* | *"bottom"* | *"left"* (float) | *"right"* (float)
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut bestimmt die Ausrichtung des IFrames als **Inline-Objekt** innerhalb der ihn umgebenden Textzeile. Die Werte `"top"`, `"middle"` und `"bottom"` bestimmen die Lage relativ zur Textzeile.

- `align="top"`
  Der obere Rand des IFrames wird an der Zeilenoberkante (d.h. nicht der Buchstabenoberkante) der aktuellen Textzeile ausgerichtet.
- `align="middle"`
  Die Mitte des IFrames wird an der Grundlinie der aktuellen Textzeile ausgerichtet.
- `align="bottom"` (Default)
  Der untere Rand des IFrames wird an der Grundlinie (d.h. nicht an den Unterlängen der Buchstaben) der aktuellen Textzeile ausgerichtet.

Die Werte `"left"` und `"right"` lösen den IFrame als **Float-Objekt** aus dem Zeilenkontext und erzwingen seine Darstellung am linken bzw. rechten Rand des umgebenden Blockelements.

- `align="left"`
  Der IFrame rückt an den linken Rand des umgebenden Blockelements, wird also auf der rechten Seite von Text umflossen.
- `align="right"`
  Der IFrame rückt an den rechten Rand des umgebenden Blockelements, wird also auf der linken Seite von Text umflossen.

Das Umfließen kann durch ein Objekt mit gesetztem `clear`-Property beendet werden (entweder über das CSS-Property `clear` oder für ein `<br>`-Element durch dessen `clear`-Attribut).

Zusätzlich zu den Werten der Spezifikation unterstützen einige Browser auch die Werte `"absbottom"`, `"absmiddle"` und `"baseline"`.

frameborder

Wert	*"0" | "1"*
Typ	Boolesche Werte in Form der Ganzzahlen 0 und 1
Verwendung	Optional

Das `frameborder`-Attribut bestimmt, ob ein Rahmen um den IFrame gezeigt wird (`frameborder="1"`, Default) oder nicht (`frameborder="0"`).

height

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `height`-Attribut bestimmt die Höhe des IFrames in Pixel oder einem Prozentwert der Höhe des Dokuments, das den IFrame beherbergt.

longdesc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Barrierefreiheit: Das `longdesc`-Attribut verweist mittels eines URL auf eine Ressource, die den im IFrame gezeigten Inhalt beschreibt. Das Attribut soll von User Agents ausgewertet werden, die keine IFrames anzeigen können.

marginheight

Wert	*number*
Typ	Eine Ganzzahl als Pixelwert
Verwendung	Optional

Das `marginheight`-Attribut legt für den oberen und unteren Dokumentrand den Abstand in Pixeln zwischen Inhalten und Rand des IFrames fest. Es hat dabei Vorrang vor dem Defaultwert.

marginwidth

Wert	*number*
Typ	Eine Ganzzahl als Pixelwert
Verwendung	Optional

Das `marginwidth`-Attribut legt für den linken und rechten Dokumentrand den Abstand in Pixeln zwischen Inhalten und Rand des IFrames fest. Es hat dabei Vorrang vor dem Defaultwert.

name

Wert	*nmtoken*
Typ	Ein Bezeichner in Form eines `NMTOKEN`
Verwendung	Optional

Das `name`-Attribut dient dazu, dem IFrame einen Bezeichner zu geben, damit er als Ziel eines Linktargets dienen kann.

scrolling

Wert	*"auto"* | *"yes"* | *"no"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `scrolling`-Attribut legt fest, ob der Inhalt des betroffenen IFrames gescrollt werden darf, und bestimmt damit gleichzeitig die Anwesenheit von Scrollbars.

- `scrolling="auto"` (Defaultwert)
  Das Scrollen wird zugelassen, die horizontalen oder vertikalen Scrollbars jedoch nur gezeigt, wenn dies durch die Größe (Breite, Höhe) des IFrameinhalts jeweils erzwungen wird.
- `scrolling="yes"`
  Das Scrollen wird zugelassen und gleichzeitig (auch falls dies anhand der Größe des IFrameinhalts nicht erforderlich ist) sowohl horizontale als auch vertikale Scrollbars gezeigt.
- `scrolling="no"`
  Das Scrollen wird untersagt und in keinem Fall Scrollbars gezeigt, auch wenn dies durch die Größe des Inhalts des IFrames erforderlich wäre.

src

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `src`-Attribut legt das Startdokument fest, das dem IFramebereich bei Aufruf des Dokuments zugeordnet und durch diesen geladen wird. Ohne gesetztes `src`-Attribut bleibt der IFramebereich leer.

width

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `width`-Attribut bestimmt die Breite des IFrames als absolute Breite in Pixel oder in Prozent des zur Verfügung stehenden Raums.

14.41.4 Proprietäre Attribute – Internet Explorer

application

Wert	*"yes" | "no"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `application`-Attribut wird im Internet Explorer in Zusammenhang mit so genannten HTML Applications (HTA) verwendet, für die reguläre Webseiten mit Endung `.hta` eingesetzt werden, die der Internet Explorer (ab Version 5.0) allerdings gesondert auswertet. Da eine HTA auf das System zugreifen kann (geringe Sicherheit), wird sie jedoch nicht von einem Webserver aus betrieben, sondern muss heruntergeladen und lokal ausgeführt werden.

Durch das `application`-Attribut kann angegeben werden, ob der IFrameinhalt als Teil der HTML Applikation betrachtet werden soll (`"yes"`) oder nicht (`"no"`, Default). Wird der Wert `"yes"` gesetzt, so können alle Scripte des IFrameinhalts mit den gleichen Privilegien auf das lokale System zugreifen wie der Rest der HTA. Auf diese Weise kann auf Scripte und Dokumente von verschiedenen Domains zugegriffen werden.

[»] *Hinweis:* Das Attribut wird ignoriert, wenn das Dokument von einem Webserver ausgeliefert wird, was bedeutet, dass Scripte nicht domainübergreifend ausgeführt werden können.

border

Wert	*number*
Typ	Ein numerischer Wert, als Pixel interpretiert
Verwendung	Optional

Das `border`-Attribut bestimmt im Internet Explorer die Breite des Rahmens um den IFrame in Pixel.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt im Internet Explorer die Rahmenfarbe des `<iframe>`-Elements. In Abwesenheit des Attributs gilt die Defaultfarbe. Der Framerahmen muss aktiviert sein, d.h. das `frameborder`-Attribut also nicht auf `"no"` oder `"0"` gesetzt sein.

hspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `hspace`-Attribut nennt für den Internet Explorer einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch links und rechts vom IFrame freizuhaltenden Raum.

noresize

Wert	*"noresize"* | *"resize"*
Typ	festgelegte Bezeichner
Verwendung	Optional

Das `noresize`-Attribut legt im Internet Explorer fest, ob der IFrame vom User in der Größe verändert werden kann. Hierfür existiert der Wert `noresize="resize"`, der die Größenveränderlichkeit ausdrücklich zulässt. Der Defaultwert ist `"noresize"`.

Anmerkung: Laut W3C-Spezifikation sollen IFrames grundsätzlich *nicht* in der Größe veränderbar sein, daher ist offiziell kein `noresize`-Attribut vorgesehen.

vspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das vspace-Attribut nennt für den Internet Explorer einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch über und unter dem IFrame freizuhaltenden Raum.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.41.5 Erläuterung zum Element

Das <iframe>-Element dient zum Einbetten eines Inlineframes in eine HTML-Seite, in dem ein externes Dokument gezeigt werden kann. Beim Element handelt es sich um einen Inlinecontainer, der im Zeilenkontext innerhalb von Blokkelementen auftreten darf.

Als Containerinhalt darf <iframe> einen Fallback-Inhalt besitzen, der alternativ dargestellt wird, wenn die Zielanwendung keine IFrames unterstützt. An dieser Stelle darf beliebiger Block- und Inlinecontent auftreten. Das Element darf allerdings auch leer bleiben.

Barrierefreiheit

Auch IFrames stellen ebenso wie Framesets eine potentielle Barriere dar und sollten daher mit Vorsicht eingesetzt werden. Im Grunde gelten alle für <frame> genannten Problematiken ebenso für <iframe> (Bookmark-, URL- und Druckprobleme, Zugänglichkeit für Suchmaschinen).

Aktuelle Screenreader können mit Frames und IFrames in der Regel umgehen, sind aber nicht in der Lage, verschiedene Frames gleichzeitig darzustellen. Stellen Sie ihnen daher ein `title`-Attribut zur Verfügung, das die im IFrame gezeigte Ressource beschreibt – dieses wird auch von textbasierten Browsern wie Lynx benötigt, um einen passenden Linktext zum Sprung zur eingebetteten Seite zu generieren. Als Teil des Alternativinhaltes ist es ansonsten empfehlenswert, einen Link zur eigentlich einzubettenden Ressource vorzusehen.

Beispiel: **[zB]**

```
<p>Hier kommt ein Inline-Frame:<br>
<iframe name="paris" title="Seite &uuml;ber Paris"
 height="200" width="350" src="paris.html">
   Der Elementinhalt wird nur von Browsern gezeigt,
   die keine Inline-Frames unterstützen.
</iframe> Das Element erzeugt keinen Zeilenumbruch, ist also auch in
diesem Sinne &quot;inline&quot;.</p>
```

Abbildung 14.49 Ein <iframe> in Mozilla

14.42 img

Das leere Inlineelement `<img>` (*image*) markiert im Sinne einer Einfügemarke die Position innerhalb einer Textzeile, an der der Browser eine externe Grafikressource in die Präsentation des HTML-Dokuments einfügen soll.

14.42.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Inline
XHTML-Modul	Image
Kontext	Überall in Block und Inlineelementen
Erlaubte Inhalte	keine (leeres Element) in HTML, XHTML 1.x XHTML 2.0: Inlineelemente und Text (#PCDATA)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<img src="bild.gif" alt="Alternativtext">
```
(HTML-Kontext)

```
<img src="bild.gif" alt="Alternativtext"/>
```
(XHTML-Kontext)

```
<img src="bild.gif">Alternativtext</img>
```
(XHTML 2.0)

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
img { display:inline; }
```

14.42.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–

Internet Explorer Events	onabort, onafterupdate, onbeforecopy, onbeforecut, onbeforepaste, onbeforeunload, onbeforeupdate, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onerror, onfilterchange, onfocus, onhelp, onload, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onrowenter, onrowexit, onselectstart, onunload

Elementspezifische Attribute

W3C	align, **alt**, border, height, hspace, ismap, longdesc, name, **src**, usemap, vspace, width
Internet Explorer	dynsrc, loop, loopdelay, lowsrc, start *Access-, Databinding-, Edit-Attribute*
Netscape	lowsrc, nosave*, suppress* * abgeschafft

14.42.3 Offizielle Attribute – W3C

align

Wert	*"top"* | *"middle"* | *"bottom"* | *"left"* (float) | *"right"* (float)
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut bestimmt primär die Ausrichtung des Bildes als Inline-Objekt innerhalb der sie umgebenden Textzeile. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

Die Werte `"top"`, `"middle"` und `"bottom"` bestimmen die Lage relativ zur Textzeile.

- `align="top"`
  Der obere Rand des Bildes wird an der Zeilenoberkante (d.h. nicht der Buchstabenoberkante) der aktuellen Textzeile ausgerichtet.
- `align="middle"`
  Die Mitte des Bildes wird an der Grundlinie der aktuellen Textzeile ausgerichtet.
- `align="bottom"` (Default)
  Der untere Rand des Bildes wird an der Grundlinie (d.h. nicht an den Unterlängen der Buchstaben) der aktuellen Textzeile ausgerichtet.

Die Werte "left" und "right" hingegen lösen das Bild als **Float-Objekt** aus dem Zeilenkontext und erzwingen seine Darstellung am linken bzw. rechten Rand des umgebenden Blockelements.

- align="left"
  Das Bild rückt an den linken Rand des umgebenden Blockelements, wird also auf der rechten Seite von Text umflossen.
- align="right"
  Das Bild rückt an den rechten Rand des umgebenden Blockelements, wird also auf der linken Seite von Text umflossen.

Das Umfließen kann durch ein Objekt mit gesetztem clear-Property beendet werden (entweder über das CSS-Property clear oder für ein
-Element durch dessen clear-Attribut).

Zusätzlich zu den Werten der Spezifikation unterstützen einige Browser auch die Werte "absbottom", "absmiddle", "baseline" und "texttop". Vom Einsatz dieser Werte muss abgeraten werden.

alt

Wert	*cdata*
Typ	Ein beliebiger Textstring als Platzhalter
Verwendung	**Obligatorisch**

Barrierefreiheit: Das obligatorische alt-Attribut enthält einen Textstring, der als Platzhalter für das Bild dienen kann, falls dieses nicht dargestellt wird. Dies ist beispielsweise bei Screenreadern der Fall, die im Zeilenkontext entsprechend den String des alt-Attribut im Textfluss auslesen sollen. Das Attribut ist *case-sensitive*.

Tipp zur Barrierefreiheit

Für Illustrationen ohne Textbezug und Spacer-Grafiken ist für das alt-Attribut sinnvollerweise ein leerer String als Wert zu setzen (»leeres alt-Attribut«). Ansonsten verlesen Screenreader eventuell den Dateinamen des Spacers oder melden pauschal das Vorhandensein einer Grafik.

Anmerkung: Das alt-Attribut dient *nicht* der eigentlichen Beschreibung des Bildinhaltes. Hierfür sind title- und longdesc-Attribute zuständig.

[»] *Hinweis:* In XHTML 2.0 *entfällt* das alt-Attribut, da das <img>-Element dort als Container definiert ist, dessen optionaler Elementinhalt die Aufgabe des alt-Attributs übernimmt.

border

Wert	*number*
Typ	Ein numerischer Wert, als Pixelwert interpretiert
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `border`-Attribut bestimmt die Breite in Pixel eines um das Bild gezeigten Rahmens, die der Breite und Höhe des Bildes zugeschlagen wird. Der Defaultwert ist abhängig vom User Agent. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

Anmerkung: Sofern das Bild als Link dient, also von einem `<a>`-Container mit `href`-Attribut umgeben ist, wird ein Rahmen von 1 Pixel Breite um das Bild gezeigt. Ist dies unerwünscht, so kann der Rahmen mittels `border="0"` deaktiviert werden.

Alternativ kann dies mittels CSS-Anweisung `style="border:0"` geschehen.

height

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `height`-Attribut bestimmt die Höhe des Bereiches, in dem die Grafik gezeigt wird, als Pixelwert oder als Prozent des verfügbaren Raums.

Die Höhenangabe mit `height` hat *Vorrang* vor der Pixelhöhe der im Bereich gezeigten Grafikdatei, kann also verwendet werden, um die Grafik beliebig zu skalieren. Wird dabei nicht gleichzeitig ein Wert für `width` übergeben, so wird die Grafik proportional (den Seitenverhältnissen folgend) skaliert, ansonsten entsprechend verzerrt.

Tipp

Die Grafikabmessungen mit `height` und `width` anzugeben, beschleunigt den Renderingvorgang, da dem Browser die Größe des frei zu haltenden Raums im Vorfeld (d.h. vor Laden der Grafikdatei) bekannt gemacht wird. Somit braucht nach dem Laden der Grafik kein erneuter Renderingdurchgang zu erfolgen.

hspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `hspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch links und rechts vom Bild freizuhaltenden Raum. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

ismap

Wert	*"ismap"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `ismap`-Attribut bestimmt, ob ein Link um das Bild als serverseitiges Imagemap dient, die Koordinaten eines Klicks in den Bildbereich also zur Auswertung an den Server geschickt werden sollen. Die Klickkoordinaten werden relativ zur Ursprungskoordinate der Graphik (deren linke obere Ecke) ausgedrückt und kommagetrennt als Querystring an den Link angehängt.

```
<a href="http://www.beispiel.com/cgi/map.pl">
   <img src="landkarte.gif" ismap="ismap" alt="Landkarte">
</a>
```

Ein Klick auf die Koordinaten `x=30` und `y=50` erzeugt somit für obiges Beispiel folgende Anfrage an den Server:

```
http://www.beispiel.com/cgi/map.pl?30,50
```

Anmerkung: Der Internet Explorer unterstützt auch die Werte `"true"` und `"false"`.

longdesc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Barrierefreiheit: Das `longdesc`-Attribut richtet sich primär an User Agents, die keine Bilder unterstützen, und wird daher von normalen Browsern nicht notwendigerweise unterstützt. Es nennt, ergänzend zu möglichen Informationen aus dem `title`-Attribut, einen URI zur Verknüpfung mit einer Ressource, die eine ausführlichere Beschreibung des Bildinhalts enthält.

[!] *Achtung:* Das `longdesc`-Attribut wird im Internet Explorer nicht unterstützt.

name

Wert	*nmtoken*
Typ	Eine Zeichenkette in Form eines NMTOKEN
Verwendung	Optional

Das name-Attribut vergibt einen Bezeichner für das Bildobjekt, damit dieses für Scripte adressierbar ist (für anonyme Bildobjekte ist eine Adressierung jedoch stets auch über die Indexziffer des Image-Arrays des Browserobjektmodells möglich).

Hinweis: Als wirkungsgleicher Ersatz sollte heutzutage das id-Attribut bevorzugt werden. **[«]**

src

Wert	*URL*
Typ	Ein URI-String
Verwendung	**Obligatorisch**

Das obligatorisch zu setzende src-Attribut nennt in Form eines URL die im Bereich des Imageobjekts zu zeigende Grafikressource.

title

Wert	*cdata*
Typ	Eine beliebige CDATA-Zeichenkette
Verwendung	Optional

Das title-Attribut dient zur Beschreibung der gezeigten Grafik, die bei Mouseover als Tooltip gezeigt wird.

usemap

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das usemap-Attribut adressiert über einen (in der Regel lokalen) URI eine mit dem Bild zu verwendende Imagemap-Definition <map>. Für eine lokale Definition besitzt der URI die Form eines Fragment-Identifiers, der den Wert des name-Attributs des <map>-Elements verwendet.

```
usemap="#map_bezeichner"
```

vspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `vspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den über und unter dem Bild freizuhaltenden Raum. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

width

Wert	*number[%]*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `width`-Attribut bestimmt die Breite des Bereiches, in dem die Grafik gezeigt wird, als Pixelwert oder als Prozent des verfügbaren Raums.

Die Breitenangabe mit `width` hat *Vorrang* vor der Pixelbreite der im Bereich gezeigten Grafikdatei, kann also verwendet werden, um die Grafik beliebig zu skalieren. Wird dabei nicht gleichzeitig ein Wert für `height` übergeben, so wird die Grafik proportional (den Seitenverhältnissen folgend) skaliert, ansonsten entsprechend verzerrt.

Tipp

Die Grafikabmessungen mit `height` und `width` anzugeben, beschleunigt den Renderingvorgang, da dem Browser die Größe des frei zu haltenden Raums im Vorfeld (d. h. vor Laden der Grafikdatei) bekannt gemacht wird. Somit braucht nach dem Laden der Grafik kein erneuter Renderingdurchgang zu erfolgen.

14.42.4 Proprietäre Attribute – Internet Explorer

controls

Wert	–
Typ	Leeres Marker-Attribut
Verwendung	Optional
Einführung	IE 3

(Historisch) Das `controls`-Attribut bestimmt im Internet Explorer, ob Kontrollmöglichkeiten zur Steuerung unterhalb eines mittels des `dynsrc`-Attributs eingebetteten AVI-Videos angezeigt werden. Das Attribut darf nicht zusammen mit den Attributen `loop` oder `start` eingesetzt werden

dynsrc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional
Einführung	ab IE 2.0

Das `dynsrc`-Attribut kann im Internet Explorer dazu eingesetzt werden, um mittels des `<img>`-Elements eine Videodatei vom Typ `.avi` als »dynamische Quelle« (*dynamic source*) einzubinden. Kann das Video nicht geladen werden (bzw. wird das Attribut nicht unterstützt), so wird alternativ das statische Bild gezeigt, das durch das `src`-Attribut bestimmt wird.

```
<img dynsrc="video.avi" src="standbild.jpg">
```

Hinweis: Die standardgerechte Alternative besteht in der Verwendung des `<object>`-Elements, bei dem ebenfalls ein statisches Bild als alternativer Inhalt bestimmt werden kann. Analog zum oberen Beispiel könnte man notieren: **[«]**

```
<object data="video.avi" type="application/avi">
   <img src="standbild.jpg">
</object>
```

loop

Wert	*number* | *"infinite"*
Typ	Eine Zahl oder ein festgelegter Bezeichner
Verwendung	Optional

Das `loop`-Attribut bestimmt im Internet Explorer zusammen mit dem `dynsrc`-Attribut durch Übergabe einer beliebigen positiven Ganzzahl die Anzahl der Durchläufe des eingebetteten AVI-Videos. Defaultwert ist `"0"` (keine Wiederholung). Der Wert `"infinite"` bewirkt eine endlose Wiederholung des Videos. Ein Synonym zu `"infinite"` stellt der Wert `"-1"` dar.

loopdelay

Wert	*number* | *"infinite"*
Typ	Eine Zahl oder ein festgelegter Bezeichner
Verwendung	Optional

Das `loopdelay`-Attribut bestimmt im Internet Explorer bei gesetztem `loop`-Attribut die Verzögerung in Millisekunden zwischen den einzelnen Wiederholungen der eingebetteten Videodatei.

lowsrc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `lowsrc`-Attribut erlaubt das vorausgehende Laden eines Vorschaubildes mit geringerer Dateigröße (sofern `width`- und `height`-Attribute gesetzt sind, auch mit kleineren Abmessungen), bevor die endgültige, durch das `src`-Attribut bestimmte Grafik geladen wird. Auf diese Weise kann subjektiv die Ladegeschwindigkeit erhöht werden. Das Attribut wird auch von Netscape und Mozilla unterstützt.

```
<img src="hauptbild.jpg" lowsrc="vorschau.jpg"
 height="100" width="130">
```

[»] *Hinweis:* Das `lowsrc`-Attribut wird nur ausgewertet, wenn sich die `src`-Grafik noch nicht im Browsercache befindet.

start

Wert	*"fileopen"* | *"mouseover"* | *"fileopen, mouseover"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `start`-Attribut bestimmt im Internet Explorer, wie das mittels eines `dynsrc`-Attribut eingebetteten AVI-Videos gestartet werden soll. Erlaubt sind die Werte `"fileopen"` (das Video wird gestartet, sobald die Videodatei geladen ist; Defaultwert) und `"mouseover"` (das Video wird gestartet, wenn der User mit der Maus über den Bereich fährt). Beide Werte können kombiniert werden, so dass mittels Mouseover ein Video nach dem ersten automatischen Durchlauf erneut gestartet werden kann.

```
<img dynsrc="video.avi" start="mouseover">
```

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.42.5 Proprietäre Attribute – Netscape

lowsrc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `lowsrc`-Attribut erlaubt das vorausgehende Laden eines Vorschaubildes mit geringerer Dateigröße (sofern `width`- und `height`-Attribute gesetzt sind, auch mit kleineren Abmessungen), bevor die endgültige, durch das `src`-Attribut bestimmte Grafik geladen wird. Auf diese Weise kann subjektiv die Ladegeschwindigkeit erhöht werden. Das Attribut wird auch im Internet Explorer unterstützt.

Hinweis: Das `lowsrc`-Attribut wird nur ausgewertet, wenn sich die `src`-Grafik noch nicht im Browsercache befindet. **[«]**

nosave

Wert	–
Typ	Leeres Marker-Attribut
Verwendung	Optional

(Historisch) Das `nosave`-Attribut wurde von Netscape Composer in `<img>`-Tags eingefügt, um anzuzeigen, dass es sich um eine lokale Bilddatei handelt, die beim Speichern der erzeugten HTML-Datei nicht erneut zusammen mit dieser gespeichert werden soll. Im Browser selbst (auch in Netscape) spielte das Attribut keine Rolle, stellte also (im Gegensatz zur häufigen Annahme) auch keine Schutzmaßnahme vor unbefugtem Speichern der Grafik dar.

suppress

Wert	*"true"* | *"false"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

(Historisch) Das `suppress`-Attribut diente in Netscape 4 dazu, das Platzhaltericon für ein (noch) nicht geladenes Bild zu unterdrücken.

14.42.6 Erläuterung zum Element

Der `<img>`-Tag ist ein Inline-Markerelement, das das Einfügen eines Bildes in den Fließtext bewirkt. Der `<img>`-Tag zählt zu den leeren Elementen. In HTML existiert daher keine Endmarke, in XHTML muss das Element in der Form `<img />` intern geschlossen werden.

Die einzubindende Grafik wird mittels des `src`-Attributs von `<img>` festgelegt, das aus diesem Grund als obligatorisch (`#REQUIRED`) definiert ist. Ebenso muss über das `alt`-Attribut ein alternativer Beschreibungstext beigegeben werden, der angezeigt wird, wenn das Bild nicht dargestellt werden kann (oder die angegebene Datei nicht verfügbar ist). Alle anderen Attribute sind optional.

Den Regeln entsprechend muss ein Bild also minimal folgendermaßen eingebunden werden:

```
<img src="beispielgrafik.jpg" alt="Beschreibungstext"/>
```

Beachten Sie, dass keine Möglichkeit vorgesehen ist, im Vorfeld den MIME-Typ der Grafik zu nennen. Der Browser muss dies aus der Dateiendung bzw. aus dem HTTP-Header entnehmen, den der Server der Datei mitgibt.

Anmerkung: In XHTML 2.0 ist das `<img>`-Element als *Container* definiert, der Text und Inlineelemente enthalten darf. Der Elementinhalt von `<img>` wird dann dargestellt, wenn die mit `src` als Quelle bezeichnete Grafikdatei nicht verfügbar ist oder nicht angezeigt werden kann. Das `alt`-Attribut entfällt.

Diese Änderung erfolgt aus Gründen der Vereinheitlichung, da <img> in XHTML 2.0 nur als Spezialfall von <object> betrachtet wird. Da in XHTML dem Inhaltsbereich jedes Elementcontainers mittels des Standardattributs src eine Grafik zugewiesen werden kann, ist das <img>-Element streng genommen überflüssig, wird aber aus Gründen der Anschaulichkeit beibehalten.

Beispiel: **[zB]**

```
<p>Ein Bild gilt als normales Zeilenelement und steht
&quot;inline:&quot;</p>

<p>Es bestimmt dabei die Zeilenhöhe <img src="bild.jpg" alt="Bild">
seiner Zeile.</p>

<p><img align="left" src="bild.jpg" alt="Bild">Das Bild kann auch
nach links (wie hier) oder rechts &quot;floaten&quot;, was entweder
mittels des <code>align</code>-Attributs oder besser mittels CSS-
Anweisungen angeordnet werden kann. Der Text umflie&szlig;t das
Bild, sofern er lang genug ist. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.</p>
```

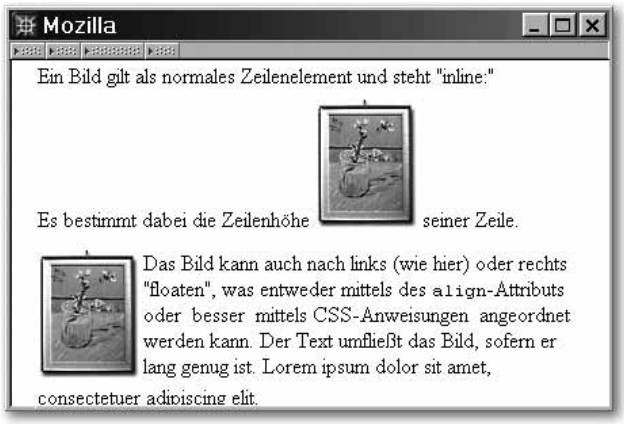

Abbildung 14.50 Ein mit <img> eingefügtes Bild im Zeilenkontext und als Float-Element

14.43 input

Das leere Element <input> dient zum Erzeugen eines Eingabefeldes innerhalb eines durch einen <form>-Container definierten Webformulars. Die Art des erzeugten Eingabefeldes wird durch das type-Attribut des Elements bestimmt.

14.43.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Inline
XHTML-Modul	Basic Forms, Forms
Kontext	Überall in Block- und Inlineelementen innerhalb eines `<form>`-Container
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01, XHTML 1.0, 1.1; XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

`<input>` (in HTML)

`<input />` (in XHTML)

Defaultrendering:

Wechselt je nach Typ des Eingabefeldes (bestimmt durch `type`-Attribut).

14.43.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	`onblur`, `onchange`, `onfocus`, `onselect`
Internet Explorer Events	`onafterupdate`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `onchange`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onerrorupdate`, `onfilterchange`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselect`, `onselectstart`

Elementspezifische Attribute

W3C	`accept`, `accesskey`, `align`, `alt`, `checked`, `disabled`, `maxlength`, `name`, `readonly`, `size`, `src`, `tabindex`, `type`, `usemap`, `value`
Internet Explorer	`autocomplete`, `dynsrc`, `lowsrc`, `notab`, `vcard_name`, *Access-, Databinding-, Edit-Attribute*
DoCoMo I-Mode	`istyle`

14.43.3 Offizielle Attribute – W3C

accept

Wert	*cdata*
Typ	ein MIME-Typ-Bezeichner
Verwendung	Optional

Das `accept`-Attribut nennt in Form eines oder mehrerer MIME-Type-Bezeichner den zu akzeptierenden Media-Typ der Eingabedaten des Eingabefeldes. Es besteht jedoch keine ausdrückliche Verpflichtung für User Agents, nur Daten vom genannten Format zuzulassen.

Das Attribut ist in Zusammenhang mit dem Inputtyp `type="file"` von Bedeutung. Hier sollen nur Dateien vom Grundtyp `image` akzeptiert werden, also `"image/png"`, `"image/gif"`, `"image/jpg"`, etc.:

```
<input type="file" name="upload" accept="image/*" />
```

Hinweis: Der Wert des `accept`-Attributs ist *nicht case-sensitive*. Die Werte `accept="IMAGE/GIF"` und `accept="image/gif"` werden daher beispielsweise als gleichwertig betrachtet. **[«]**

accesskey

Wert	*[a-z][1–0]*#*
Typ	Alphanumerisches Zeichen, sowie * und #; nicht *case-sensitive*.
Verwendung	Optional

Barrierefreiheit: Das Attribut `accesskey` bestimmt im Sinne der Accessibility ein einzelnes alphanumerisches Tastaturzeichen, das im Sinne eines Keyboard-Short-

cuts das Eingabefeld in den Fokus nimmt – unter Windows bei gleichzeitig gedrückter Alt-Taste[31], unter Mac OS mit Ctrl-Taste.

```
<input type="text" name="suchen" accesskey="s" />
```

Das Eingabefeld im oberen Beispiel wird mit Alt+S ausgewählt.

[»] *Hinweis:* Auf Webseiten für mobile Endgeräte sollten als Accesskeys lediglich die Ziffern 0–9 verwendet werden. Des Weiteren sind theoretisch auch * (Asterisk) und # (Hash-Symbol) gestattet, wobei (geräteabhängig) die durch die entsprechenden Tasten gesteuerte Seitenscrollfunktion Vorrang vor den Accesskeys besitzt.

align

Wert	*"top"* | *"middle"* | *"bottom"* | *"left"* (float) | *"right"* (float)
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut bestimmt die Ausrichtung eines Grafik-Buttons mit `type="image"` als Inline-Objekt innerhalb der ihn umgebenden Textzeile. Die Werte `"top"`, `"middle"` und `"bottom"` bestimmen die Lage relativ zur Textzeile.

- `align="top"`
  Der obere Rand des Buttons wird an der Zeilenoberkante (d.h. nicht der Buchstabenoberkante) der aktuellen Textzeile ausgerichtet.
- `align="middle"`
  Die Mitte des Buttons wird an der Grundlinie der aktuellen Textzeile ausgerichtet.
- `align="bottom"` (Default)
  Der untere Rand des Buttons wird an der Grundlinie (d.h. nicht an den Unterlängen der Buchstaben) der aktuellen Textzeile ausgerichtet.

Die Werte `"left"` und `"right"` hingegen lösen den Button als **Float-Objekt** aus dem Zeilenkontext und erzwingen seine Darstellung am linken bzw. rechten Rand des umgebenden Blockelements.

- `align="left"`
  Der Button rückt an den linken Rand des umgebenden Blockelements, wird also auf der rechten Seite von Text umflossen.

31 Unter Windows in Opera-Browsern statt mit Alt mit gleichzeitig gedrückter ⇧- und Esc-Taste. Hintergedanke ist, den Tastaturzugang zum Browsermenü per Alt-Taste nicht zu behindern. Dieser bleibt jedoch bei einzelner Betätigung der Alt-Taste auch bei den anderen Browsern unbeschränkt möglich.

- `align="right"`
  Der Button rückt an den rechten Rand des umgebenden Blockelements, wird also auf der linken Seite von Text umflossen.

Das Umfließen kann durch ein Objekt mit gesetztem `clear`-Property beendet werden (entweder über das CSS-Property `clear` oder für ein `<br>`-Element durch dessen `clear`-Attribut).

alt

Wert	*cdata*
Typ	Ein beliebiger Textstring als Platzhalter
Verwendung	Optional

Barrierefreiheit: Das `alt`-Attribut enthält einen Textstring, der als **Platzhalter** für einen Grafik-Button mit `type="image"` dient, falls dieser nicht dargestellt wird. Dies ist beispielsweise bei Screenreadern der Fall, wo dann der String des `alt`-Attributs im Textfluss ausgelesen wird.

checked

Wert	*"checked"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `checked`-Attribut bestimmt, ob ein Inputelement des Typs Checkbox oder Radiobutton beim Laden des Formulars vorgewählt ist. Defaultzustand ist nicht vorgewählt. Innerhalb eines Radiobutton-Arrays darf nur genau ein `checked`-Attribut gesetzt werden, innerhalb eines Checkbox-Arrays ist dies beliebig.

Anmerkung: Wird in einem Radiobutton-Array nirgends ein `checked`-Attribut vergeben, so soll der User Agent den ersten Button des Arrays vorwählen. Da hierauf in der Praxis kein Verlass ist, sollte besser explizit ein `checked`-Attribut gesetzt werden.

disabled

Wert	*"disabled"*
Typ	Festgelegter Bezeichner
Verwendung	Optional

Das `disabled`-Attribut deaktiviert ein Formulareingabeelement und unterdrückt die Versendung dessen Werts bei Abschicken der Formulardaten. Gleichzeitig

wird durch das Attribut verhindert, dass der User den Zustand des Elements ändert (durch Werteingabe oder Anklicken). Ein deaktiviertes Eingabeelement wird durch Ausgrauung präsentiert.

Anmerkung: Der Internet Explorer unterstützt auch die Werte `"true"` (Feld deaktiviert) und `"false"` (nicht deaktiviert; Default).

ismap

Wert	*"ismap"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `ismap`-Attribut bestimmt, ob ein Grafik-Button mit `type="image"` als serverseitige Imagemap dient, die Koordinaten eines Klicks in den Buttonbereich also zur Auswertung an den Server geschickt werden sollen. Die Klickkoordinaten werden relativ zur Ursprungskoordinate der Graphik (deren linke obere Ecke) ausgedrückt und zusammen mit dem *control name* des Inputs verschickt.

maxlength

Wert	*number*
Typ	Eine Ganzzahl; maximale Länge des Eingabestrings
Verwendung	Optional

Das `maxlength`-Attribut gibt für Eingabefelder vom Typ »Text« oder »Passwort« die maximal akzeptierte Länge des Eingabestrings an und verhindert Eingaben des Nutzers, die diese Länge überschreiten. Der Wert kann geringer sein, als die durch das `size`-Attribut festgelegte Breite des Eingabefeldes. Der Defaultwert bei Abwesenheit des Attributs ist `"unlimited"` (unbegrenzt).

name

Wert	*nmtoken*
Typ	Ein Bezeichner in Form eines NMTOKEN
Verwendung	Optional

Das `name`-Attribut legt den Bezeichner (*control name*) des Eingabefeldes innerhalb eines Formulars fest. Der Bezeichner findet bei der Datenübertragung als `name`-Part eines »`name-value`«-Pärchens Verwendung. Der Attributwert ist nicht *case-sensitive*, daher werden beispielsweise `name="FELDNAME"` und `name="feldname"` als identisch betrachtet.

Der Gültigkeitsbereich von *control names* beschränkt sich auf das beherbergende Formular – es dürfen *control names* (im Gegensatz zu IDs) in einem Dokument daher mehrfach erscheinen, solange sie zu verschiedenen Formularen gehören. Zur Bildung eines **Buttonarrays** *muss* ein `name`-Bezeichner innerhalb eines Formulars mehrfach vergeben werden.

Hinweis: Der Wert eines Feldes ohne `name`-Attribut wird beim Datenversand nicht berücksichtigt. Es ist üblich, Submit- und Reset-Buttons nicht eigens zu benennen, es sei denn, es muss zwischen mehreren solcher Buttons unterschieden werden können. **[«]**

readonly

Wert	*"readonly"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `readonly`-Attribut kann eingesetzt werden, um eine Änderung eines Vorgabewertes durch den Nutzer zu unterbinden. Die Anwendung erstreckt sich im Wesentlichen auf Texteingabefelder, die scriptgesteuert Werte ausgeben, die der Nutzer zur Kontrolle betrachten können, aber nicht ändern soll.

Achtung: Das `readonly`-Attribut ist auch auf Checkboxen oder Radiobuttons anwendbar, wobei aber nicht die Anwahl des Elements verhindert wird, sondern lediglich (für den User nicht sichtbar) die Änderung des übertragenen Wertes. **[!]**

Hinweis: Die Werte von als `"readonly"` gekennzeichneten Elementen werden, anders als im Falle von `"disabled"`, in jedem Fall übertragen. **[«]**

size

Wert	*number*
Typ	Eine Ganzzahl als Breite in Zeichen interpretiert
Verwendung	Optional

Das `size`-Attribut gibt für Eingabefelder vom Typ »Text« oder »Passwort« die sichtbare Breite des Feldes in Zeichen an. Die absolute Breite des Feldes ist dabei abhängig von der für das Element geltenden Schriftgröße. Der Defaultwert für `size` ist 20.

Achtung: Für die Beschränkung der Länge der einzugebenden Zeichenkette ist das Attribut `maxlength` zuständig – `size` hat hierauf keinen Einfluss. **[!]**

src

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `src`-Attribut nennt in Form eines URL die für einen grafischen Submit-Button vom Typ »Image« zu zeigende Grafikressource. (In der Kombination mit `type="image"` ist das `src`-Attribut *obligatorisch*. Zusätzlich sollte das `alt`-Attribut eingesetzt werden, um einen Alternativtext zur Verfügung zu stellen.)

tabindex

Wert	*Integerwert*
Typ	Positive Ganzzahl
Verwendung	Optional

Barrierefreiheit: Die an das Attribut `tabindex` übergebene Ganzzahl legt in aufsteigender Richtung die Position des Formularelements in der Tabulatorwahl-Reihenfolge (⇆-Taste) fest.

Die Tab-Reihenfolge gestaltet sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes `tabindex`-Attribut in Quelltextreihenfolge. (Zu den aktivierbaren Elementen gehören allgemein neben `<input>` auch `<a>`, `<area>`, `<button>`, `<object>`, `<select>` und `<textarea>`.)

[»] *Hinweis:* Die Tab-Reihenfolge enthält alle aktivierbaren Elemente gemeinsam, also auch Links. Für Links und Formularelemente sollten daher niemals gleiche Tabindex-Werte gesetzt werden.

type

Wert	*"text"* | *"password"* | *"checkbox"* | *"radio"* | *"submit"* | *"reset"* | *"file"* | *"hidden"* | *"image"* | *"button"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `type`-Attribut bestimmt die Art des zu erzeugenden Eingabefeldes. Der Defaultwert in Abwesenheit des Attributs ist `"text"`. Es existieren insgesamt zehn definierte Typen von Eingabefeldern.

Anmerkung: Die »Web Forms 2.0«-Spezifikation der WHAT WG schlägt sechs weitere Input-Typen vor: `"number"` (nur numerische Eingaben), `"time"` (Zeitangaben oder Zeitstempel), `"date"` (Datumswerte), `"range"` (Eingabe innerhalb eines vorgegebenen Bereichs), `"email"` (E-Mail-Adresse), `"url"` (URI-String).

Des Weiteren wird ebendort ein `pattern`-Attribut vorgeschlagen, das Eingaben in Felder vom Typ `"text"`, `"password"`, `"email"` und `"url"` mittels regulärer Ausdrücke einschränken kann.

usemap

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `usemap`-Attribut ist einsetzbar in Zusammenhang mit einem grafischen Submit-Button vom Typ »Image«. Es adressiert über einen (in der Regel lokalen) URI eine mit dem Bild zu verwendende Imagemap-Definition `<map>`. Für eine lokale Definition besitzt der URI die Form eines Fragment-Identifiers, der den Wert des `name`-Attributs des `<map>`-Elements verwendet.

```
usemap="#map_bezeichner"
```

value

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Das `value`-Attribut bestimmt den Vorgabewert eines Eingabefeldes.

- Im Falle eines **Textfeldes** wird dieser Wert beim Laden oder Zurücksetzen des Formulars in das Feld geschrieben, kann jedoch anschließend vom Nutzer gelöscht oder verändert werden.
- Für **Checkboxen** und **Radiobuttons** bestimmt `value` den (für User unveränderbaren) Wert, der für ein angewähltes Element beim Versenden des Formulars übertragen wird.
- Für Eingabeelemente in **Buttonform** (Typ »Button«, »Submit« oder »Reset«) wird mittels `value` die Buttonaufschrift festgelegt (der Wert wird allerdings auch übertragen, sofern der Button ein `name`-Attribut besitzt).

14.43.4 Proprietäre Attribute – Internet Explorer

autocomplete

Wert	*"on"* | *"off"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	IE5

Autocomplete: Das `autocomplete`-Attribut bestimmt im Internet Explorer, ob für ein Eingabefeld die Autovervollständigen-Funktion aktiviert werden soll oder nicht. Defaultwert ist `"off"` (Autocomplete inaktiv).

Anmerkung: Das `autocomplete`-Attribut wird auch in Mozilla/Firefox sowie in Opera 9.0 unterstützt. Es ist außerdem Bestandteil der Web Forms 2.0 Spezifikation der WHAT WG.

dynsrc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `dynsrc`-Attribut kann für einen Input vom Typ »Image« im Internet Explorer dazu eingesetzt werden, um eine Videodatei vom Typ `.avi` als »dynamische Quelle« (*dynamic source*) einzubinden. Kann das Video nicht geladen werden (bzw. wird das Attribut nicht unterstützt), so wird alternativ das statische Bild als Button verwendet, das durch das `src`-Attribut bestimmt wird.

lowsrc

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `lowsrc`-Attribut erlaubt im Internet Explorer für einen Input vom Typ »Image« das vorausgehende Laden eines Vorschaubildes mit geringerer Dateigröße (sofern `width`- und `height`-Attribute gesetzt sind, auch mit kleineren Abmessungen), bevor die endgültige, durch das `src`-Attribut bestimmte Grafik geladen wird. Auf diese Weise kann subjektiv die Ladegeschwindigkeit erhöht werden.

notab

Wert	–
Typ	leeres Marker-Attribut
Verwendung	Optional
Gültigkeit	IE3 (abgeschafft seit IE4)

(Historisch) Das `notab`-Attribut diente dazu, das Objekt aus der Tab-Reihenfolge des Internet Explorers zu entfernen. Für das gewünschte Ergebnis kann dort auch das `tabindex`-Attribut auf den Wert –1 gesetzt werden.

vcard_name

Wert	*cdata*
Typ	ein Bezeichner eines vCard-Datenfeldes
Verwendung	Optional

Autocomplete: Das Attribut `vcard_name` nennt für ein Eingabefeld vom Typ Text oder Passwort das Datenfeld eines im standardisierten vCard-Format[32] nach RFC 2426 gespeicherten Datensatzes, der für den Browser unter dem aktiven Profil zugänglich ist. Unter einer vCard ist eine vorwiegend in Zusammenhang mit E-Mail-Adressen eingesetzte »elektronische Visitenkarte« zu verstehen.

Beispiel: Dieses Eingabefeld bezieht seinen Wert aus dem Feld EMAIL einer mit dem aktuellen Profil verknüpften vCard: **[zB]**

```
<input type="text" name="mail" vcard_name="vCard.email">
```

Hat diese etwa folgende Form

```
BEGIN:VCARD
VERSION:3.0
N:Tester;Thorsten
FN:Thorsten Tester
ORG:Example
URL:http://www.example.org/
EMAIL;INTERNET:thorsten.tester@example.org
END:VCARD
```

So bietet Autocomplete die Mail-Adresse `thorsten.tester@example.org` als Feldinhalt an. Gibt der User an dieser Stelle eine andere E-Mail-Adresse ein, so wird diese automatisch innerhalb der gleichen Webdomain für andere, mit dem selben vCard-Feld verknüpfte Inputs der Vorgabeliste hinzugefügt.

32 vCard Standard, siehe RFC 2426: http://www.ietf.org/rfc/rfc2426.txt.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`dataformatas = "text" | "localized-text" | "HTML"`

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.43.5 Proprietäre Attribute – DoCoMo I-Mode

istyle

Wert	*"1" | "2" | "3" | "4"*
Typ	number
Verwendung	Optional
Einführung	iMode HTML 2.0

Mit dem `istyle`-Attribut wird der Eingabestil (alphabetisch, numerisch) eines Eingabefelds vom Typ »Text« in i-Mode kompatiblen Browsern festgelegt. Als Attributwerte sind die Zahlen von 1 bis 4 zugelassen. Von den Vorgaben abweichende Eingabewerte bewirken den Defaultstyle »Full size kana«.

- `istyle="1"` (Default)
  Full-size kana
- `istyle="2"`
  Half-size kana
- `istyle="3"`
  Alphabetic
- `istyle="4"`
  Numeric

Hinweis: Das istyle-Attribut ist nicht anwendbar auf Eingabefelder vom Typ "password", da diese bei I-Mode-Geräten grundsätzlich und unveränderlich den Eingabestil »numerisch« (Mode 4) besitzen. [«]

14.43.6 Erläuterung zum Element

Das Element <input> erzeugt Eingabefelder verschiedenen Typs, die innerhalb eines Webformulars benötigt werden. Es kommt daher nur innerhalb des <form>-Containers vor.

Gemäß der *strict*-DTDs darf es jedoch nicht als unmittelbares Kindelement von <form> erscheinen, sondern es muss stets ein Blockelement (wie <fieldset>, <p>, oder eine Liste beliebiger Art) dazwischengeschaltet sein. Bei <input> handelt es sich um ein leeres Element: In HTML ist daher eine Endmarke verboten, in XHTML muss der Tag intern als <input/> geschlossen sein.

Rein formal sind alle Attribute von <input> optional, wobei das type-Attribut den Defaultwert "text" besitzt. Folgende Angabe wäre also gültig und erzeugt ein Texteingabefeld in Defaultbreite (20 Zeichen):

```
<!-- Textinput in XHTML: -->
<input/>
```

Das name-Attribut ist zwar ebenfalls als optional (#IMPLIED) definiert, *muss* aber jedem Eingabefeld als *control name* zugeordnet werden, um den Datenversand zu ermöglichen (Bildung von *name-value*-Pärchen). Eine *Ausnahme* hiervon bilden lediglich Inputs in Form von Submit- und Reset-Buttons (hier macht sich ein name-Attribut in der Regel eher störend bemerkbar). Korrekter ist daher als Minimalangabe:

```
<!-- Textinput in XHTML: -->
<input name="Texteingabe" />
```

Das name-Attribut in XHTML

Das name-Attribut ist in XHTML zugunsten von id abgeschafft und wird lediglich für Formularelemente wie <button>, <input>, <select> und <textarea> weiterhin beibehalten.

Typen von Eingabefeldern

In HTML und XHTML sind zehn Typen von Inputs definiert. Der meistbenötigte Typ `"text"` ist als Defaultwert definiert. Alle anderen werden über das `type`-Attribut festgelegt.

- `type="text"` (Default)
  Erzeugt ein einfaches **Texteingabefeld**, dessen Breite (in Zeichen) durch das `size`-Attribut und dessen maximal akzeptierte Eingabestringlänge durch das `maxlength`-Attribut bestimmt werden. Alle Texteingabefelder eines Formulars müssen verschiedene `name`-Attributwerte besitzen.
- `type="password"`
  Erzeugt ein **Passworteingabefeld**, bei dem eingegebene Zeichen in der Darstellung durch Sterne (Asterisken) als Platzhalter ersetzt werden. Die interne Zwischenspeicherung (*current value* des Objekts) und Übertragung des eingegebenen Wertes erfolgt jedoch in Klartext. Das Passwortfeld gleicht ansonsten dem Eingabefeld vom Typ `"text"`.
- `type="checkbox"`
  Erzeugt ein Boolesches Eingabefeld in Form einer **Checkbox**. Mehrere Checkboxen mit gleichem `name`-Atributwert (*control name*) bilden ein Array, in dem beliebig viele Checkboxen (keine bis alle) ausgewählt werden können. Die Werte aller gewählten Eingabefelder werden einem Property zugeordnet und gemeinsam übertragen. Eine vorzuwählende Checkbox wird mit dem Attribut `checked` markiert.
- `type="radio"`
  Erzeugt ein Boolesches Eingabefeld in Form eines **Radiobuttons**. Mehrere Radiobuttons mit gleichem `name`-Atributwert (*control name*) bilden ein Array, in dem genau ein Radiobutton ausgewählt werden kann. Ein vorzuwählender Radiobutton wird mit dem Attribut `checked` markiert – ansonsten soll der User Agent den ersten Radiobutton des Arrays vorwählen. Nur der Wert des gewählten Eingabefelds wird übertragen (für eine korrekte Funktion müssen die Radiobuttons unterschiedliche *values* besitzen).
- `type="submit"`
  Erzeugt einen **Submit-Button**, der auf Klick einen Submit-Event erzeugt und den Dateninhalt des Formulars an die Adresse schickt, die das `action`-Attribut des umgebenden `<form>`-Containers bestimmt. Die Aufschrift des Submit-Buttons lautet defaultmäßig (in Abhängigkeit von der Spracheinstellung) »Submit«, kann aber über das `value`-Attribut des Elements beliebig festgelegt werden. Sofern der Button ein `name`-Attribut besitzt, wird der `value`-Wert, also die Buttonaufschrift, zusammen mit den Formulardaten verschickt. Soll dies nicht erfolgen, kann `name` weggelassen werden.

Hinweis: Ein Formular darf mehrere Submit-Buttons bzw. Elemente mit Submit-Funktion besitzen. Es kann festgestellt werden, welcher Button betätigt wurde, da nur das `name`-Attribut des aktivierten Buttons zusammen mit seinem `value` übertragen wird. **[«]**

- `type="image"`
  Bindet über das `src`-Attribut eine Grafik ein, die als Schaltfläche für den Submitvorgang dient. Aus Accessibilitygründen sollte ein **Grafikbutton** stets zusätzlich ein `alt`-Attribut mit passendem Wert erhalten.

  Ein Input dieses Typs kann zusätzlich mittels der Attribute `usemap` bzw. `ismap` als client- oder serverseitiges Imagemap gekennzeichnet werden. Im ersten Fall werden die Definitionen des per `usemap` adressierten `<map>`-Containers verwendet, im zweiten die Klickkoordinaten in Verbindung mit dem Bezeichner (*control name*) des Grafikbuttons übertragen.

- `type="reset"`
  Erzeugt einen **Reset-Button**, der vorgenommene Eingaben des Nutzers löscht und die Eingabefelder auf die Vorgabewerte zurücksetzt. Die Aufschrift des Reset-Buttons lautet defaultmäßig (in Abhängigkeit von der Spracheinstellung) »Reset«, kann aber über das `value`-Attribut des Elements beliebig festgelegt werden.

- `type="button"`
  Erzeugt einen **neutralen Button** ohne festgelegte Wirkung, dessen Beschriftung durch sein `value`-Attribut bestimmt wird. Dieser Buttontyp wird gewöhnlich eingesetzt, um Scripte zu steuern.

- `type="hidden"`
  Erzeugt ein so genanntes **verstecktes Feld**, dessen Wert vom Nutzer nicht unmittelbar geändert werden kann. Ein verstecktes Feld kann mittels seines `value`-Attributs mit einem Wert vorbelegt werden. Ansonsten können dynamisch erzeugte (z. B. berechnete) Werte per Script dort abgelegt werden. Dies ist beispielsweise bei kaskadierten Formularen von Nutzen, die im Vorfeld eingegebene Werte weiterreichen müssen.

- `type="file"`
  Erzeugt ein **Dateiupload-Feld**, mit dem der Nutzer eine Datei durch eine Browse-Funktion auf seinem eigenen Rechner auswählen und beim Verschicken der Formulardaten mitversenden kann. Für das `enctype`-Attribut des `<form>`-Containers muss `enctype="multipart/form-data"` gesetzt werden, damit ein Dateiupload ermöglicht wird.

  Das `accept`-Attribut kann den Media-Typ der hochgeladenen Datei einschränken. Hier sollen nur Dateien vom Grundtyp `image` akzeptiert werden, also `"image/png"`, `"image/gif"`, `"image/jpg"` etc.:

```
<input type="file" name="upload" accept="image/*" />
```

[!] *Achtung:* Eine Größenbeschränkbarkeit der hochgeladenen Datei in Bytes mittels des `maxlength`-Attributs ist zwar vorgesehen, aber nur in HTML 3.2 definiert. Eine Unterstützung dieser Funktionalität durch Browser sollte daher nicht als gegeben betrachtet werden.

Hier wird die Dateigröße auf 50 kB beschränkt:

```
<input type="file" name="upload" maxlength="50000" />
```

[zB] *Beispiel:* In folgendem Formular sind die verschiedenen Inputtypen vertreten.

```
<form method="post" action="bla.cgi" name="mein_formular">
<input type="hidden" name="versteckt" value="geheim"/>
<input type="text" name="textfeld1"/>
<input type="password" name="passwort"/>

<input type="checkbox" name="ch_box" value="checkbox1"/>
<input type="checkbox" name="ch_box" value="checkbox2"/>
<input type="checkbox" name="ch_box" value="checkbox3"/>
<input type="radio" name="r_button" value="radiobutton1"/>
<input type="radio" name="r_button" value="radiobutton2"/>
<input type="radio" name="r_button" value="radiobutton3"/>

<select name="selectbox">
    <option>Das ist eine Selectbox</option>
    <option>... und eine Auswahl</option>
</select>

<select name="select" size="2">
    <option value="no1">erstes Listenelement</option>
    <option value="no2">zweites Listenelement</option>
    <option value="no3">drittes Listenelement</option>
</select>

<textarea name="textfield" cols="15" rows="5">
Defaulttext
</textarea>

<input type="file" name="file"/>

<input type="button" name="mein_button" value="Button"/>
<input type="submit" name="mein_submit" value="Submit"/>
<input type="reset" name="mein_reset" value="Reset"/>
</form>
```

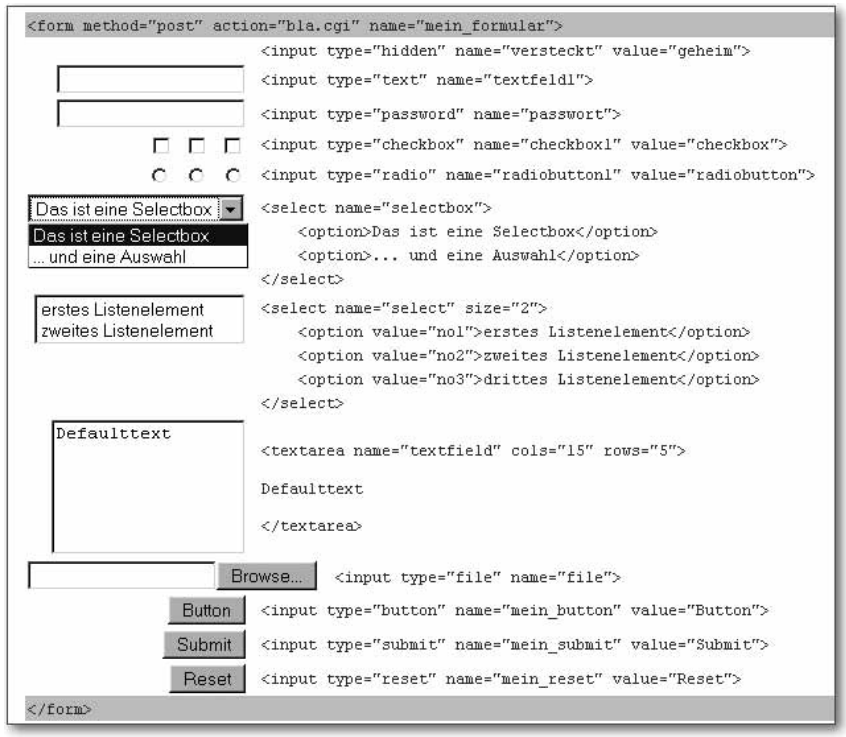

Abbildung 14.51 Inputs eines Formulars und ihr Quellcode

14.44 ins

Logische Auszeichnung: Das `<ins>`-Element markiert Einfügungen und Ergänzungen des Dokumentinhalts im Sinne einer Änderungsmarkierung.

14.44.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer/Blockcontainer (je nach Kontext)
XHTML-Modul	Edit
Kontext	Überall in Block und Inlineelementen (auch in `<body>`)
Erlaubte Inhalte	Block- und Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1 **abgeschafft in:** XHTML 2.0
Mobil-Spezifikationen	– *Nicht enthalten* in HTML 4.0 mobile, XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<ins> ... </ins>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
ins   { display: inline;
        text-decoration: underline;    /* vorgesehen */
        }
```

14.44.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`cite`, `datetime`
Internet Explorer	*Access-, Edit-Attribute*

14.44.3 Offizielle Attribute – W3C

cite

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `cite`-Attribut enthält einen URI als Quelle für Angabe von Gründen für die Änderungen bzw. über den Urheber.

datetime

Wert	*cdata*
Typ	Ein String in Form eines Zeitstempelwertes oder Datums
Verwendung	Optional

Das `datetime`-Attribut enthält Angaben über den Zeitpunkt der Änderung in Form eines Datums- oder Zeitstempelwertes. Das Attribut ist *case-sensitive*.

```
<!-- eingefügt am 25. Oktober 2006: -->
<ins datetime="2006-10-25"> ... </ins>

<!-- eingefügt am 25. Oktober 2006 um 12:00: -->
<ins datetime="2006-10-25T12:00"> ... </ins>
```

14.44.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.44.5 Erläuterung zum Element

Das Element dient der Markierung von Textpassagen, die im Rahmen einer Überarbeitung eingefügt wurden. Eingefügte Textstellen werden üblicherweise durch Unterstrich markiert dargestellt. Mittels des `datetime`-Attributs kann der Zeitpunkt der Änderung angegeben werden, wobei ein Datums- oder ein Zeitstempelwert übergeben werden kann. Das `cite`-Attribut nennt optional den Urheber der Änderung oder verweist auf den Änderungsgrund (beispielsweise über einen URI, der dann jedoch nur dokumentarische und nicht funktionale Bedeutung besitzt).

Das Element fungiert gewöhnlich als **Inlinecontainer** innerhalb eines Blockcontainers. Ebenso ist es jedoch möglich, Quelltextpassagen aus mehreren Blockelementen als Einfügung auszuzeichnen, weshalb `<ins>` ausdrücklich auch auf Blocklevel-Ebene erscheinen darf (als unmittelbarer Inhalt des `<body>`-Elements) und damit quasi die Funktion eines **Blockelements** übernimmt.

[zB] *Beispiel:*

```
<p>Dies ist Text, wobei einige Passagen
   <del>gel&ouml;scht</del>
   <ins>eingef&uuml;gt</ins>
   wurden.</p>
```

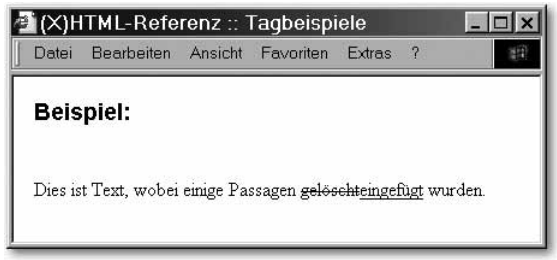

Abbildung 14.52 Präsentation von <ins> und <del> im Internet Explorer

Anmerkung: Das Element `<ins>` als explizite Einfügemarke *entfällt* in XHTML 2.0. Seine Aufgabe kann (besser) von dem neuen Standardattribut `edit` übernommen werden, dessen erlaubte Werte `"inserted"`, `"deleted"`, `"changed"` und `"moved"` es ermöglichen, jedem Elementcontainer Informationen über inhaltliche Veränderungen mitzugeben.

Das Attribut `datetime`, das den Zeitpunkt der Änderung in Form eines Zeitstempels vermerkt, kann nun ebenfalls jedem Element zugewiesen werden. Beide Attribute sind Teil der Edit-Collection von XHTML 2.0 und gehören als Teil der Common Collection zu den Standardattributen.

14.45 isindex

Deprecated: Das Element `<isindex>` markiert das Dokument als durchsuchbar und blendet ein Eingabefeld für Suchanfragen ein, die nach Betätigen der ↵-Taste an den Server geschickt werden.

14.45.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element
XHTML-Modul	Legacy
Kontext	Im `<head>`- oder `<body>`-Container
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0/4.01 (tr/fs) XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0 / 4.01 strict XHTML 1.0 strict, 1.1., 2.0
Mobil-Spezifikationen	cHTML (deprecated) **abgeschafft** in HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

`<isindex>` (HTML)

`<isindex />` (XHTML)

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

–

14.45.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `title`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `onfocus`, `onreadystatechange`, `onresize`

Elementspezifische Attribute

W3C	`prompt`
Internet Explorer	`action`, *Access-*, *Edit-Attribute*
Netscape	`action`, `target`

14.45.3 Offizielle Attribute – W3C

prompt

Wert	*cdata*
Typ	Ein beliebiger Textstring
Verwendung	Optional

Das `prompt`-Attribut erlaubt die Festlegung des links neben dem Inputfeld stehenden Eingabeaufforderungstexts. Das Attribut ist *case-sensitive*.

Der Defaulttext lautet (abhängig von den Spracheinstellungen des Browsers):

»This is a searchable index. Enter search keywords:«

oder

»Sie können diesen Index durchsuchen, indem Sie den Suchbegriff eingeben:«

[»] *Hinweis:* Soll *kein Text* neben dem Eingabefeld erscheinen, muss als Attributwert für Mozilla-Browser mindestens ein Leerzeichen übergeben werden, da sonst der Defaulttext erscheint. Im Internet Explorer genügt der leere String.

14.45.4 Proprietäre Attribute – Internet Explorer

action

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `action`-Attribut legt die Zieladresse fest, an die die Anfrage geschickt werden soll, um verarbeitet zu werden.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.45.5 Proprietäre Attribute – Netscape:

action

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `action`-Attribut legt die Zieladresse fest, an die die Anfrage geschickt werden soll. Es kann sich dabei um eine serverseitige Anwendung (CGI-Script) oder ähnliches handeln.

target

Wert	*"_blank"* | *"_self"* | *"_parent"* | *"_top"* | *cdata*
Typ	Ein Frame- oder Fenstername
Verwendung	Optional

Das `target`-Attribut bestimmt das Zielfenster oder den Zielframe, in den die Ergebnisse der Anfrage ausgegeben werden sollen. Existiert kein entsprechend benanntes Fenster, so wird ein neues Browserfenster unter dem übergebenen Bezeichner geöffnet. Ist kein `target`-Attribut gesetzt, so werden die Ergebnisse im gleichen Fenster ausgegeben, aus dem die Suchanfrage kommt.

Hinweis: Das Attribut ist *case-sensitive*. **[«]**

14.45.6 Erläuterung zum Element

Das Element `<isindex>` erzeugt ein Texteingabefeld, das der Nutzer für Suchanfragen verwenden kann. Die eingegebene Zeichenkette wird nach Betätigen der

↵-Taste als Querystring an den Basis-URL der aktuellen Seite (bzw. den durch das `action`-Attribut des Elements übergebene URL) angehängt und durch den Webserver ausgewertet. Dieser schickt die Antwort zum Browser zurück, der sie im gleichen Fenster darstellt, sofern nicht im `target`-Attribut ein anderes Zielfenster angegeben wurde.

Gibt der Nutzer beispielsweise den String `"rosen tulpen nelken"` ein und der Basis-URI der Seite ist `www.example.com`, so wird folgende Query ausgeführt:

```
http://www.example.com/?rosen+tulpen+nelken
```

... wobei Leerzeichen innerhalb des Strings entsprechend des normalen URL-Encodings durch Pluszeichen maskiert werden. Da keine Möglichkeit besteht, einen erlaubten Zeichensatz zu spezifizieren, sind die erlaubten Zeichen in der Praxis auf ASCII bzw. Latin-1 beschränkt.

[»] *Hinweis:* Wird das Element im Dokumentkopf eingesetzt, sollte es nur einmal verwendet werden. Innerhalb des Bodys dürfen mehrere `<isindex>`-Elemente (beispielsweise mit verschiedenen `action`-Attributwerten) erscheinen. Generell wird von der Verwendung zugunsten flexiblerer HTML-Formulare abgeraten.

[zB] *Beispiel:*

```
<isindex>

<isindex prompt="Sie k&ouml;nnen diese Seite durchsuchen: ">

<isindex prompt="In Paris suchen: " action="paris.html">

<isindex prompt="Extern in Paris suchen: " action="paris.html"
target="_blank">
```

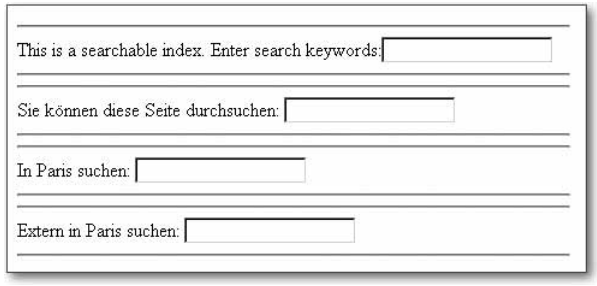

Abbildung 14.53 Das Element <isindex> in Mozilla

Anmerkung: Der Internet Explorer interpretiert das `<isindex>`-Element intern als `<input name="isindex">` innerhalb eines `<form>`-Containers.

14.46 kbd

Logische Auszeichnung: Das Element `<kbd>` markiert seinen Elementinhalt als Tastatureingabe (*keyboard*) und präsentiert ihn in nicht-proportionaler Schrift (Monospace).

14.46.1 Elementeigenschaften

Einführung:	HTML 1.0
Elementtyp:	Inlinecontainer
XHTML-Modul:	Text
Kontext:	Überall in Block- und Inlineelementen
Erlaubte Inhalte:	Inlineelemente und Text
Offizielle W3C Spezifikationen:	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile **nicht enthalten** in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<kbd> ... </kbd>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
kbd    { font-family: monospace }
```

14.46.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–

Internet Explorer Events	onbeforecut, onbeforepaste, onblur, oncontextmenu, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	– (keine)
Internet Explorer	Access-, Edit-Attribute

14.46.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.46.4 Erläuterung zum Element

Das Element `<kbd>` ist ein Inlineelement, das der logischen Auszeichnung des Elementinhalts als (meist vom Nutzer) vorzunehmende Tastatureingabe oder zur Markierung eines Tastatur-Shortcuts dient. Obwohl der Inhalt üblicherweise durch einen Monospace-Font (beispielsweise Courier) dargestellt wird, zählt man `<kbd>` nicht zu den Präsentationselementen.

Anmerkung: Die Darstellung von `<kbd>` entspricht den Elementen `<code>`, `<samp>` und `<tt>`, wobei Letzteres jedoch reine Präsentationsbedeutung (»physische Auszeichnung«) besitzt.

[zB] *Beispiel:*

```
<p>Dr&uuml;cken Sie nun die Taste <kbd>ESC</kbd>.
   Wenn Sie Gl&uuml;ck haben ...</p>
```

14.47 label

Das <label>-Element kann einem Formulareingabefeld über eine IDREF-Beziehung eine klicksensitive Beschriftung zuordnen, die das Eingabefeld auf Mausklick in den Fokus nimmt.

14.47.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Basic Forms, Forms
Kontext	Überall in Block- und Inlineelementen innerhalb eines <form>-Containers
Erlaubte Inhalte	Inlineelemente (mit Ausnahme von <label> selbst) und Text (#PCDATA)
Offizielle W3C Spezifikationen	HTML 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile **abgeschafft** in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<label> ... </label>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
label   {display:inline;}
```

14.47.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	onblur, onfocus

Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, oncontextmenu, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	accesskey, for
Internet Explorer	Access-, Databinding-, Edit-Attribute

14.47.3 Offizielle Attribute – W3C

accesskey

Wert	*[a-z][1–0]*#*
Typ	Alphanumerisches Zeichen sowie * und #; nicht *case-sensitive*.
Verwendung	Optional

Barrierefreiheit: Das Attribut accesskey bestimmt im Sinne der Accessibility ein einzelnes alphanumerisches Tastaturzeichen, das im Sinne eines Keyboard-Shortcuts das Eingabeelement in den Fokus nimmt, dem das <label>-Element zugewiesen ist – unter Windows bei gleichzeitig gedrückter Alt-Taste, unter Mac OS mit Ctrl-Taste.

```
<label accesskey="s" for="suchen">
   <input type="text" id="suchen" />
</label>
```

Das Eingabefeld im oberen Beispiel wird mit Alt+S in den Fokus genommen.

for

Wert	*idref*
Typ	Eine Zeichenkette in Form eines ID-Bezeichners
Verwendung	Optional

Barrierefreiheit: Das for-Attribut gibt im Sinne einer IDREF-Beziehung den Identifier des Elements an, dem das Label zugewiesen wird. Der Inhaltsbereich des Labels dient als vergrößerte Klickfläche, um das Eingabeelement in den Fokus zu nehmen. Das Attribut ist *case-sensitive*.

14.47.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`dataformatas = "text" | "localized-text" | "HTML"`

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.47.5 Erläuterung zum Element

Das `<label>`-Element steht im Zentrum barrierefreier Formulare. Erstens dient es dazu, mittels einer `ID-IDREF`-Beziehung den Zusammenhang zwischen einem Eingabefeld und seiner Beschriftung zu verdeutlichen, zum anderen vergrößert es die anklickbare Fläche, um den Fokus auf ein gewünschtes Eingabefeld zu setzen. Zusätzlich kann genau wie für das Input-Element auch ein Accesskey (im Internet Explorer auch ein Tabindex) gesetzt werden.

Zur logischen Verknüpfung beider Elemente benötigt das Eingabeelement denselben `ID`-Wert, den das Label für sein `for`-Attribut erhält:

```
<label for="username">Username: </label>
   <input type="text" id="username" name="username" />
```

Label und Input müssen für korrekte Funktion zwar nicht unmittelbar benachbart sein, laut BITV sollte dies jedoch dennoch stets der Fall sein, damit dem Nutzer der Bezug auch über die Optik des Formulars ersichtlich ist (BITV 10.2)

Auch Eingabeelemente eines Buttonarrays können einzeln mit Labels versehen werden. Für die Inputs können allerdings nicht gleiche `name`- und `id`-Werte vergeben werden, da die Elemente eines Arrays den gleichen `name`-Wert benötigen. Da der `id`-Wert nicht für den Versand der Daten zuständig ist, kann hier ohne Weiteres ein individueller `ID` gesetzt werden.

```
<label for="button_1">Button 1: </label>
   <input type="radio" id="button_1" name="array_1"/><br/>
<label for="button_2">Button 2: </label>
   <input type="radio" id="button_2" name="array_1"/><br/>
<label for="button_3">Button 3: </label>
   <input type="radio" id="button_3" name="array_1"/>
```

[»] *Hinweis:* Der Einsatz von `<label>` für das Styling des Formularlayouts ist in Zusammenhang mit dem `<form>`-Container und dem `<dt>`-Container beschrieben.

[zB] *Beispiel:* Als Experiment für die Übertragung des Klick-Events vom Label auf »seinen« Input dient folgendes kleines Beispiel. Am Input befindet sich lediglich ein Eventhandler, der den Klick-Event auffängt; es spielt keine Rolle, ob auf den Input direkt oder auf das mit diesem verknüpfte Label geklickt wird.

```
<form action="">
<p><input onclick="alert(this.value)" id="rosen" value="Rosen"
name="blume" type="radio">
<label for="rosen">Hier klicken für Rosen</label></p>
 ...
<p><input id="wahl" name="wahl" type="text">
<label onclick="alert('Textfeld wird fokussiert.')" for="wahl">
... oder klicken Sie hier, um selbst zu w&auml;hlen!</label></p>
</form>
```

Abbildung 14.54 Das Label überträgt seinen Klick-Event auf den Radiobutton.

14.48 legend

Das `<legend>`-Element definiert eine Beschriftung für Formularelemente, die eine durch einen `<fieldset>`-Container zusammengefasste logische Gruppe bilden.

14.48.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Forms
Kontext	Nur in `<fieldset>`
Erlaubte Inhalte	Inline-Elemente und Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 4.0, 4.01, XHTML 1.0, 1.1; **abgeschafft in:** XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile *nicht enthalten* in XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<legend> ... </legend>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
legend      {display:inline; }
```

14.48.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–

Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize

Elementspezifische Attribute

W3C	accesskey, align
Internet Explorer	Access-, Edit-Attribute

14.48.3 Offizielle Attribute – W3C

accesskey

Wert	*[a-z][1–0]*#*
Typ	Alphanumerisches Zeichen, sowie * und #; nicht *case-sensitive*.
Verwendung	Optional

Barrierefreiheit: Das Attribut accesskey bestimmt im Sinne der Accessibility ein einzelnes alphanumerisches Tastaturzeichen, das im Sinne eines Keyboard-Shortcuts die Elementgruppe (bzw. den ersten Input der Gruppe) in den Fokus nimmt – unter Windows bei gleichzeitig gedrückter Alt-Taste, unter Mac OS mit Ctrl-Taste.

align

Wert	*"top"* | *"bottom"* | *"left"* | *"right"* (| *"center"*)
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das align-Attribut richtet den Elementinhalt, der als Beschriftung einer Gruppe von Formularelementen dient, gegenüber der Gruppe aus. Als offizielle Werte sind "top", "bottom", "left" und "right" gestattet, wenn auch browserseitig größtenteils nicht im Sinne der Spezifikation umgesetzt.

In der Praxis wird der (inoffizielle) Wert "center" hingegen von den meisten Browsern unterstützt (Ausnahme: Opera, der das align-Attribut nicht berücksichtigt.).

- align="top" (Default)
  Gruppenüberschrift über der Gruppe. (Praxis: oben links)

- `align="bottom"`
  Gruppenüberschrift unter der Gruppe. (Praxis: oben links)
- `align="left"`
  Gruppenüberschrift links neben der Gruppe. (Praxis: oben links)
- `align="right"`
  Gruppenüberschrift rechts neben der Gruppe. (Praxis: oben rechts)
- `align="center"` (inoffiziell)
  Im Internet Explorer eingeführt. Gruppenüberschrift erscheint zentriert über der Gruppe. (Da dieser Wert nicht spezifiziert ist, ist seine Unterstützung nicht dauerhaft gewährleistet.)

14.48.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.48.5 Erläuterung zum Element

Das Element `<legend>` dient dazu, einer Eingabefeldgruppe eine optionale Überschrift zuzuweisen. Es muss dafür als erstes Kindelement im `<fieldset>`-Container erscheinen.

Das Element stellt einen Container dar; die Endmarke `</legend>` ist vorgeschrieben. Als Inhalte dürfen Text und Inlineelemente erscheinen, auch Hyperlinks, Grafiken, Objekte sind somit neben Text und Textformatierungselementen zugelassen.

Beispiel: **[zB]**

```
...
<p>Legend align="right":</p>
<fieldset>
<legend align="right">Lieferung</legend>
<input name="lieferung" type="radio">per Express
```

```
<input name="lieferung" type="radio">per UPS
<input name="lieferung" type="radio">pers&ouml;nlich
</fieldset>
<p>Legend align="top":</p>
<fieldset>
<legend align="top">Lieferung</legend>
<input name="lieferung" type="radio">per Express
<input name="lieferung" type="radio">per UPS
<input name="lieferung" type="radio">pers&ouml;nlich
</fieldset>
...
```

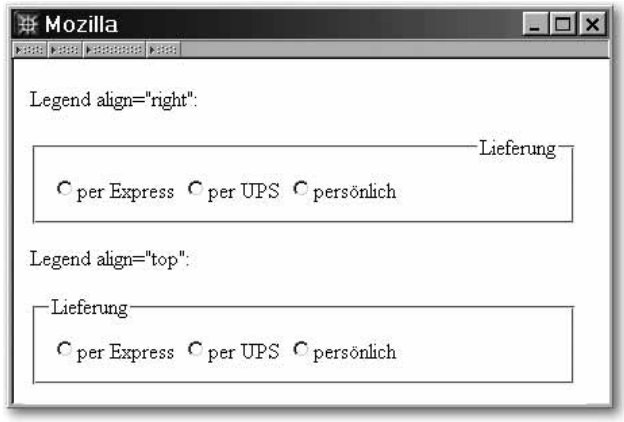

Abbildung 14.55 Die Überschrift (legend) einer Eingabefeldgruppe

14.49 li

Das `<li>`-Element kennzeichnet ein Listenelement innerhalb einer Verzeichnisliste `<dir>`, einer Menüliste `<menu>`, einer geordneten Liste `<ol>` oder einer ungeordneten Liste `<ul>`.

14.49.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	List
Kontext	Innerhalb von Listen vom Typ `<dir>`, `<menu>`, `<ol>` oder `<ul>`
Erlaubte Inhalte	Block- und Inlineelemente und Text (in `<dir>` und `<menu>` nur Inlineelemente und Text)

Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<li> ... </li>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
li        { display: list-item }
```

14.49.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	type, value
Internet Explorer	Access-, Edit-Attribute

14.49.3 Offizielle Attribute – W3C

type

Wert	*"disc"* | *"circle"* | *"square"* | *"1"* | *"a"* | *"A"* | *"i"* | *"I"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `type`-Attribut bestimmt die Art des Bullets bzw. Nummerierungszeichen des Listenelements `<li>`.

Der Defaultwert ergibt sich aus der Art der umgebenden Liste. Für `<ul>` ist dies ein Bullet vom Typ `"disc"`, für `<ol>` eine Nummerierung mit arabischen Ziffern. (Für `<dir>` und `<menu>` ist kein Default festgelegt; üblich ist `"disc"`).

- `type="disc"`
  erzeugt ein Bullet in Form einer schwarzen Scheibe.
- `type="circle"`
  erzeugt ein Bullet in Form eines Kreises.
- `type="square"`
  erzeugt ein Bullet in Form eines schwarzen Quadrats.
- `type="1"`
  erzeugt eine Nummerierung mit arabischen Ziffern.
- `type="a"`
  erzeugt eine Nummerierung mit Kleinbuchstaben.
- `type="A"`
  erzeugt eine Nummerierung mit Großbuchstaben.
- `type="i"`
  erzeugt eine Nummerierung mit kleinen römischen Ziffern.
- `type="I"`
  erzeugt eine Nummerierung mit großen römischen Ziffern.

Anmerkung: Der Wert des Attributs hat stets Vorrang vor der defaultmäßig oder dort ebenfalls per `type`-Attribut vorgegebenen Bulletform des umgebenden Listencontainers `<dir>`, `<menu>`, `<ol>` oder `<ul>`.

So ist auch die Zuordnung eines neutralen Bullets (*disc*, *circle* oder *square*) innerhalb einer geordneten Liste `<ol>` oder die eines Nummerierungszeichen (1, a, A, i, I) innerhalb einer ungeordneten Liste `<ul>` möglich. In letzterem Fall ergibt sich die angezeigte Nummerierung aus der Position des Listenelements in der Liste, als ob es sich bei dieser um eine geordnete Liste handelte.

Achtung: Im Internet Explorer setzt der Versuch, ein artfremdes Bullet festzulegen, das entsprechende Listenelement auf den *Defaulttyp* der umgebenden Liste (*nicht* etwa auf den dort per `type`-Attribut vorgegebenen Typ). **[!]**

value

Wert	*number*
Typ	Ganzzahl als Wert des Aufzählungszeichens
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `value`-Attribut bestimmt mittels einer Ganzzahl den Nummerierungswert eines `<li>`-Elements innerhalb einer geordneten Liste. Die Art des ausgegebenen Nummerierungszeichens ist abhängig vom `type`-Attribut des Listenelements selbst oder der umgebenden Liste vom Typ `<ol>`.

Hier wird dem Listenelement die Ziffer 3 vorangestellt:

```
<ol>
<li value="3">Ein Listenelement</li>
<li>Ein Listenelement</li>
</ol>
```

Hier ist es der Kleinbuchstabe c:

```
<ol type="a">
<li value="3">Ein Listenelement</li>
<li>Ein Listenelement</li>
</ol>
```

Die Zählung der Liste wird für den Rest der Liste vom gesetzten Wert aus weitergeführt. Das zweite Element der ersten Liste des obigen Beispiels erhält daher die 4, das der zweiten Liste den Buchstaben b. (In dieser Beziehung ähnelt das `value`-Attribut dem `start`-Attribut der Listencontainer.)

Achtung: Die Übergabe eines falschen Zeichens (beispielsweise eines Buchstabens) bewirkt im Internet Explorer und in Opera eine *Neu-Nummerierung* der Liste ab dem betreffenden Listenelement. Mozilla/Firefox hingegen ignorieren einen entsprechenden Wert. **[!]**

14.49.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.49.5 Erläuterung zum Element

Das Element dient als Container eines Listenelements innerhalb einer Liste der Form `<ul>`, `<ol>`, `<dir>` oder `<menu>` (die beiden letztgenannten Typen gelten jedoch als »deprecated« und sollten nicht mehr eingesetzt werden). Das `<li>`-Element tritt nur als Kindelement der soeben genannten Listentypen auf.

Das `<li>`-Element ist ein Container. Die Endmarke darf in HTML weggelassen werden (verkürzbar), nicht jedoch in XHTML. Grundsätzlich ist empfehlenswert, sie stets einzusetzen. Als Inhalte sind Block- und Inlineelemente sowie Text zugelassen. Im Rahmen verschachtelter Listen können `<li>`-Container auch andere Listen vom Typ `<ul>` oder `<ol>` enthalten. Eine Verschachtelung des Containers mit sich selbst ist hingegen nicht gestattet.

```
<ol>
   <li>Tomaten</li>
   <li>Basilikum</li>
</ol>

<ol type="a">
   <li>Rosen</li>
   <li>Tulpen</li>
   <li>Nelken</li>
</ol>

<ul>
   <li type="disc">Tomaten</li>
   <li type="circle">Basilikum</li>
</ul>
```

Abbildung 14.56 Listenelemente in geordneten und ungeordneten Listen

Listenelemente in einer `<menu>`-Liste werden angezeigt wie in einer `<ul>`-Liste:

```
<p>Menuliste:</p>
<menu>
   <li>Home</li>
   <li>&Uuml;ber uns</li>
   <li>Kontakt</li>
</menu>
```

Abbildung 14.57 Listenelemente in einer Menüliste

14.50 link

Das `<link>`-Element definiert die Rolle des aktuellen Dokuments zu einer anderen Datei in Form einer logischen Beziehung. In der Praxis wird das Element hauptsächlich zur Einbindung von Stylesheet-Dateien eingesetzt.

14.50.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Headkontext
XHTML-Modul	Link
Kontext	Im Dokumentkopf `<head>`
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile iMode XHTML 1.0, 1.1

Syntaxbeispiel:

`<link>` (HTML)

`<link />` (XHTML)

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

`link` { display: none; }

14.50.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onerror`, `onload`, `onreadystatechange`

Elementspezifische Attribute

W3C	`charset`, `href`, `hreflang`, `media`, `rel`, `rev`, `target`, `type`, `urn`[33]
Internet Explorer	`disabled`, `name`

14.50.3 Offizielle Attribute – W3C

charset

Wert	*encoding-bezeichner*
Typ	Ein registrierter Bezeichner für Character-Encodings[34].
Verwendung	Optional

Das `charset`-Attribut teilt dem Browser das Encoding-Schema der über das `<link>`-Element verknüpften Ressource mit. Default-Encoding ist ISO 8859–1 (ISO Latin 1), eine Untergruppe von ISO 8859.

href

Wert	*URL*
Typ	Absoluter oder relativer URL der verlinkten Ressource
Verwendung	Optional

Das `href`-Attribut nennt der URI der Ressource, die durch das `<link>`-Element eingebunden werden soll.

hreflang

Wert	*lang-bezeichner*
Typ	Ein Sprachenbezeichner nach RFC 1766[35].
Verwendung	Optional

Das Attribut `hreflang` nennt mittels einer Kurzbezeichnung nach RFC 1766 die Sprache der durch das `<link>`-Element adressierten Ressource.

media

Wert	*cdata*
Typ	Ein Medientyp-Bezeichner
Verwendung	Optional

33 Abgeschafft in HTML 3.2.
34 Siehe http://www.iana.org/assignments/character-sets.
35 RFC 1766: siehe www.ietf.org/rfc/rfc1766.txt.

Das `media`-Attribut bezeichnet in Form einer kommagetrennten Liste den oder die Medientyp(en) der verlinkten Ressource.

In der Praxis wird dies für verknüpfte Stylesheets eingesetzt, um die Anwendung des Stylesheets für ein bestimmtes Ausgabemedium zu kennzeichnen. Gängige Werte sind `screen`, `print`, `projection`, `handheld`, `aural` und `all`.

- `all`
  Das Stylesheet wird für alle Medientypen angewendet, also grundsätzlich *gemeinsam* mit allen anderen verknüpften Stylesheets.
- `aural`
  Barrierefreiheit: Das Stylesheet wird für Sprachausgabe angewendet.
- `braille`
  Barrierefreiheit: Das Stylesheet wird für Ausgabe mittels eines Braille-Lesegeräts angewendet.
- `embossed`
  Barrierefreiheit: Das Stylesheet dient der Steuerung von Druckern für Braille-Ausgabe.
- `handheld`
  Das Stylesheet wird für Ausgabe auf Mobiltelefonen, Organizern und vergleichbaren Ausgabegeräten eingesetzt.
- `print`
  Das Stylesheet wird für Druckausgabe eingesetzt.
- `projection`
  Das Stylesheet wird für Projektion mittels Beamer angewendet.
- **`screen`** (Default)
  Das Stylesheet wird für Bildschirmausgabe angewendet. Dieser Wert gilt defaultmäßig, wenn kein `media`-Attribut verwendet wird.
- `tty`
  Das Stylesheet wird bei Ausgabe mit textorientierten Browsern angewendet (eigentlich im Hinblick auf Fernschreiber eingeführt).
- `tv`
  Das Stylesheet wird für Ausgabe auf Fernsehgeräten angewendet, die sich technisch (Auflösung, Zeilenzahl etc.) von Computerbildschirmen unterscheiden.

[»] *Hinweis:* Die Einbindung von Stylesheets für bestimmte Medientypen kann auch mittels der `@import`-Syntax vorgenommen werden.

Anstelle von

```
<link rel="stylesheet" media="handheld"
 href="mobile.css" type="text/css" />
```

kann man auch notieren

```
<style type="text/css">
   @import url(mobile.css) handheld;
</style>
```

rel

Wert	*rel-bezeichner*
Typ	Vordefinierter Bezeichner für Link-Relationen.
Verwendung	Optional

Ein vordefinierter Relation-Bezeichner oder eine durch Leerzeichen getrennte Liste aus mehreren dieser Bezeichner. Das `rel`-Attribut beschreibt die Beziehung (*relation*) der verlinkten Ressource zum verlinkenden Dokument. Wiederholbare `rel`-Werte können bei mehreren `<link>`-Elementen innerhalb desselben Dokuments verwendet werden, bei allen anderen wird nur das jeweils letzte `<link>`-Element mit dem entsprechenden Attributwert berücksichtigt.

Hinweis: Generell ist bei wiederholbaren Ressourcen die Verwendung des `title`-Attributs anzuraten, da diese in irgendeiner Form im Browsermenü oder einer Navigationsleiste erscheinen können. In Zusammenhang mit Stylesheets entscheidet die An- oder Abwesenheit eines Titels über die Persistenz oder Gruppenzugehörigkeit des Stylesheets. **[«]**

- `alternate`
  Wiederholbar: Link zu einer Alternativversion des aktuellen Dokuments, beispielsweise einer Übersetzung (in diesem Fall dient das `lang`- oder `xml:lang`-Attribut zur Beschreibung der Sprache). Wird gleichzeitig das `media`-Attribut eingesetzt, so wird eine alternative Version für ein entsprechendes Ausgabemedium bezeichnet.
- `alternate stylesheet`
  Wiederholbar: Link zu einem alternativen Stylesheet, das der User zur Darstellung hinzuziehen kann. Diese Relation kann im Sinne der Accessibility eingesetzt werden, um für Nutzer mit Handicaps andere Darstellungsmöglichkeiten anzubieten (große Schrift, anderes Farbschema).

  Hinweis: Gleichzeitig wird das `title`-Attribut benötigt, um dem Stylesheet einen Titel zu geben, der im Browsermenü anwählbar ist. Mehrere alternative **[«]**

Stylesheets können hierzu auch durch Vergabe eines identischen Titels gruppiert und gemeinsam aktiviert und deaktiviert werden.

- `appendix`
  Link zum Anhang des Dokuments oder der Dokumentenfolge.
- `author`
  Link zu Angaben über den Autor oder die Autoren des Dokuments (Kontakt, Adresse, persönliche Angaben etc.) oder ein Verweis auf ein Impressum.
- `bookmark`
  Wiederholbar: Link zu einem beliebigen Zielpunkt innerhalb des Dokuments oder einer Dokumentenfolge.
- `chapter`
  Wiederholbar: Link zu einem Kapitelbeginn innerhalb des Dokuments oder einer Dokumentenfolge.
- `contents`
  Link zum Inhaltsverzeichnis des Dokuments oder einer Dokumentenfolge
- `copyright`
  Copyrightangaben zum Dokument
- `first`
  Die in einer logischen Reihenfolge erste Seite
- `glossary`
  Link zum Glossar
- `help`
  Aufruf einer Hilfefunktion
- `index`
  Ein Stichwortverzeichnis zum Dokument oder einer Dokumentenfolge
- `last`
  Die in einer logischen Reihenfolge letzte Seite
- `next`
  Die in einer logischen Reihenfolge folgende Seite; kann eingesetzt werden, um die Folgeseite in den Cache vorzuladen (wird von WebTV unterstützt).
- `prev`
  Die in einer logischen Reihenfolge vorangehende Seite
- `search`
  Aufruf einer Suchfunktion
- `section`
  Wiederholbar: Ein Abschnitt (*section*) des Dokuments oder einer Dokumentenfolge.

- `subsection`
  Wiederholbar: Ein Unterabschnitt (*subsection*) des Dokuments oder einer Dokumentenfolge.
- `start`
  Die Startseite der Website (oder einer Dokumentenfolge)
- `stylesheet`
  Wiederholbar: Ein Stylesheet, das mit dem Dokument verknüpft werden soll. Dessen Typ muss über das `type`-Attribut genannt werden.

  Das Stylesheet gilt als »persistent« (uneingeschränkt anzuwenden), wenn *kein* `title`-Attribut verwendet wird. Ein alternatives Stylesheet (*alternate stylesheet*) wird *zusätzlich* zu diesem Stylesheet angewendet.

  Hinweis: Wird gleichzeitig ein `title`-Attribut verwendet, so gilt das Stylesheet als »preferred«. Es ist von vornherein aktiv, kann aber durch Anwahl eines alternativen Stylesheets vom Nutzer deaktiviert werden. Mehrere Stylesheets können hierzu auch durch Vergabe eines identischen Titels gruppiert und gemeinsam aktiviert und deaktiviert werden. **[«]**
- `toc`
  Link zum Inhaltsverzeichnis (`table of contents`). Synonym zu `contents`, das allerdings nicht generell unterstützt wird.
- `top`
  Die Startseite der Website (oder einer Dokumentenfolge)
- `up`
  Eine übergeordnete Kapitelseite

Hinweis: Das `title`-Attribut dient in diesem Zusammenhang der Beschreibung der Ressource (z. B. für Stylesheets bei Mozilla als Eintrag im Menü **View > Use Style**). Gegebenenfalls erscheint der Titel auch in der generierten Buttonleiste des Browsers, sofern diese Funktionalität unterstützt wird. **[«]**

rev

Wert	*rel-Bezeichner*
Typ	Vordefinierter Bezeichner für Link-Relationen.
Verwendung	Optional

Enthält einen vordefinierten Relation-Bezeichner oder eine durch Leerzeichen getrennte Liste aus mehreren dieser Bezeichner.

Das `rev`-Attribut dient in Umkehrung zum `rel`-Attribut dazu, die Beziehung des verlinkenden Dokument zur verlinkten Ressource zu beschreiben. Als vordefinierte Werte gelten ebenfalls die für das `rel`-Attribut angegebenen Werte.

target

Wert	*nmtoken*
Typ	Nametoken als Frame- oder Fensterbezeichner
Verwendung	Optional

Ein freigewählter Bezeichner eines (existierenden) Browserfensterobjekts oder im Framesetkontext einer der vordefinierten Bezeichner `"_self"`, `"_parent"`, `"_top"` oder `"_blank"`. Ein explizit im `<link>`-Element gesetztes Target hat Vorrang vor einem mit Hilfe des `<base>`-Elements und dessen `target`-Attribut gesetzten globalen Target.

type

Wert	*cdata*
Typ	Ein MIME-Typ-Bezeichner
Verwendung	Optional

Das Attribut `type` nennt den MIME-Typ (Multipurpose Internet Mail Extension) der Ressource, auf die das `<link>`-Element zielt. Die Angabe besteht aus der Angabe eines Hauptmedientyps und einem Subtyps, die durch Slash getrennt sind – z. B. `"text/html"`, `"text/css"`, `"image/gif"`, `"application/xml"` sowie eine Reihe weiterer (siehe RFC 2045, 2046 und 2077).

urn

Wert	*cdata*
Typ	Ein String in Form eines URN
Verwendung	Optional
Gültigkeit	**abgeschafft** in HTML 3.2

(Historisch) Das `urn`-Attribut sollte gemäß HTML 1.0/2.0 die Verwendung von URNs (Universal Resource Names) zum Einbinden von Stylesheets ermöglichen. Die Verwendung wurde jedoch nie unterstützt.

14.50.4 Proprietäre Attribute – Internet Explorer:

disabled

Wert	*"disabled" | "true" | "false"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das disabled-Attribut dient im Internet Explorer dazu, eine Ressource zwar zu verknüpfen, diese Verknüpfung aber unberücksichtigt zu lassen. Dieses Attribut richtet sich im Wesentlichen auf Stylesheets, die mittels der proprietären JScript-Funktion setActiveStyleSheet() aktiviert werden können. Hierbei muss der Name des Stylesheets übergeben werden. Der Defaultwert ist "false".

name

Wert	*NMTOKEN*
Typ	Ein Bezeichner in Form eines Nametoken
Verwendung	Optional

Das name-Attribut dient im Internet Explorer der Benennung einer verlinkten Ressource. Es wird eingesetzt, um deaktivierte Stylesheets mittels der Funktion setActiveStyleSheet() aktivieren zu können.

14.50.5 Erläuterung zum Element

Das leere Element <link> tritt ausschließlich im <head>-Kontext auf. Es dient zur Definition einer Verknüpfung des aktuellen Dokuments mit einer externen Ressource, wobei mit rel- oder rev-Attribut die Art der Beziehung zwischen Dokument und Ressource beschrieben werden kann.

Einbinden von Stylesheets

Die gebräuchlichste Verwendung von <link> ist das Einbinden externer Stylesheetdateien, hierfür muss rel="stylesheet" gesetzt sein:

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

Es können mehrere, sich addierende Stylesheets mit einem Dokument verknüpft werden. Ansonsten können diesen Stylesheets über das media-Attribut verschiedene Ausgabemedien zugewiesen werden. Die wichtigsten sind media="print" für Druckausgabe oder media="screen" für Bildschirmausgabe.

```
<link rel="stylesheet" type="text/css" href="screen.css"
 media="screen" />
<link rel="stylesheet" type="text/css" href="print.css"
 media="print" />
```

Alternative Stylesheets

Mittels rel="stylesheet" eingebundene Stylesheets sind grundsätzlich stets für alle jeweils entsprechenden Ausgabemedien aktiviert (ist für das Haupt-Style-

sheet kein `media`-Attribut gesetzt, so tritt bei Druckausgabe ein mit `media="print"` gekennzeichnetes Stylesheet nur *zusätzlich* in Kraft!).

Wollen Sie ein Stylesheet als »alternatives« Stylesheet einbinden, so muss es mit `rel="alternate stylesheet"` gekennzeichnet sein. Es *muss* gleichzeitig ein `title`-Attribut erhalten, das in der Menüauswahl des Browsers den Eintrag kennzeichnet, mit dem das Stylesheet aktiviert wird:

```
<link rel="alternate stylesheet" type="text/css"
 title="Anzeige in gro&szlig;er Schrift"  href="grosseschrift.css"
/>
```

Generierte Navigationsleisten mit <link>

In Opera kann auf Grundlage der Angaben von Linkelementen ein zusätzliches Navigationselement in Form einer Buttonleiste permanent eingeblendet werden (**Ansicht • Symbolleisten • Navigationsleiste**; Abbildung 14.58, oben).

Die Reihenfolge und Anzahl der Buttons sind festgelegt, die Beschriftung abhängig von der Spracheinstellung. Wiederholbare Elemente und dokumentinterne Navigation (`chapter`, `subsection` etc.) werden nicht berücksichtigt. Die Linkelemente werden mit dem korrespondierenden Button über das `rel`-Attribut verknüpft, bei Klick wird die über das `href`-Attribut bestimmte Ressource aufgerufen. Existiert für einen Button kein `<link>`-Element mit korrespondierendem `rel`-Attribut, so ist der entsprechende Button ausgegraut.

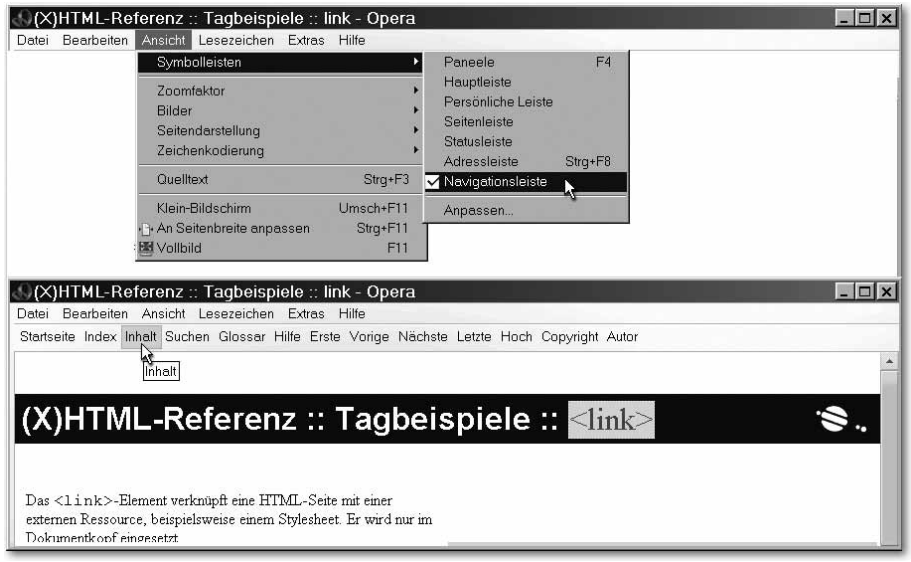

Abbildung 14.58 Link-Buttonleiste in Opera (unten)

In Netscape, ICab und Mozilla ist die Einblendung einer Zusatznavigation ebenfalls möglich (**View • Show/Hide • Site Navigation Bar**), wobei der Nutzer die Wahl hat, die Leiste permanent oder nur dann einzublenden, wenn Linkelemente vorhanden sind, die diese Funktionalität erfordern (Abbildung 14.59).

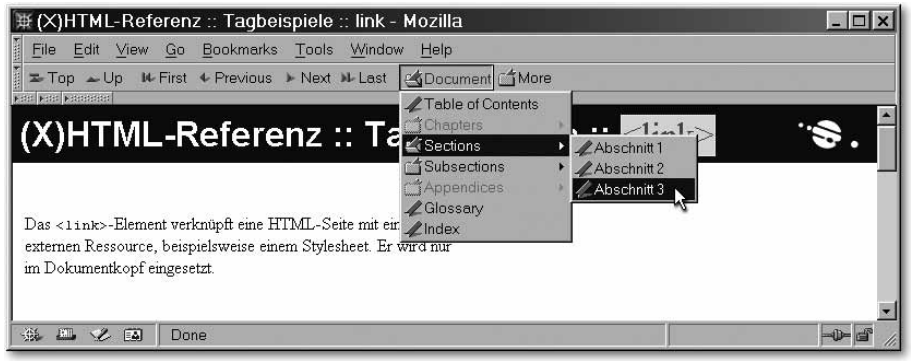

Abbildung 14.59 Link-Buttonleiste in Mozilla

In diesem Falle werden auch wiederholbare Elemente berücksichtigt, beispielsweise die `rel`-Typen `section`, `subsection`, die über die als Pulldown-Menüs gestalteten Buttons **Document** und **More** zugänglich sind. Nicht verwendete Funktionen sind ausgegraut.

Im Internet Explorer wird die Generierung einer Zusatznavigation nicht unterstützt. Ab Internet Explorer 7.0 kann immerhin `rel="search"` verwendet werden, um eine Suchmöglichkeit anzubieten, die vom Nutzer aus der Suchliste des Browsers heraus aufgerufen werden kann. Das `type`-Attribut *muss* hierfür den Wert `"application/opensearchdescription+xml"` besitzen:

```
<link title="Suche mit ..." rel="search"
   type="application/opensearchdescription+xml"
   href="http://www.beispielsuchmaschine.de/descr.xml" />
```

Bei der Zielressource *muss* es sich um ein XML-Dokument in Form einer **OpenSearch Description**[36] handeln, die die Suchfunktion bzw. den für die Suche einzusetzenden Provider beschreibt. Der Nutzer kann der Einbindung dieser Suchfunktion zustimmen oder sie ablehnen.

Achtung: Es handelt sich hier also *nicht* um einen einfachen Verweis auf eine Webseite, die beispielsweise eine siteinterne Suche anbietet. **[!]**

36 Spezifikation zur OpenSearch Description: siehe http://opensearch.a9.com/spec/1.1/description/.

14.51 listing

Historisch: Der `<LISTING>`-Container diente zur Darstellung längerer Quellcodebeispiele innerhalb eines HTML-Dokuments.

14.51.1 Elementeigenschaften

Einführung	HTML Level 0
Elementtyp	Blockcontainer
XHTML-Modul	– (abgeschafft)
Kontext	erlaubt in `<blockquote>`, `<body>`, `<dd>`, `<form>`, `<li>`
Erlaubte Inhalte	Text (`CDATA`)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, HTML 2.0 (deprecated), HTML 3.2 (deprecated) **abgeschafft** in HTML 4.0/4.01; XHTML (alle)
Mobil-Spezifikationen	– (nirgends spezifiziert)

Syntaxbeispiel:

```
<LISTING> ... </LISTING>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
LISTING    { font-family: monospace;
             white-space: pre;
        /*   ggfs. kleinere Darstellung:  */
        /*   font-size: smaller;          */
             }
```

14.51.2 Attribute (Übersicht)

Standardattribute

HTML	–
XHTML	–

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–

Internet Explorer Events	–

Elementspezifische Attribute

W3C	–

14.51.3 Erläuterung zum Element

Seit HTML 4.0 abgeschafft: Ähnlich wie der <pre>-Container stellt der <LISTING>-Container seinen Inhalt in Schrift konstanter Laufweite dar, wobei eine minimale Zeilenbreite von 132 Zeichen vorgesehen war (sonst wird gewöhnlich von 80 Zeichen/Zeile ausgegangen). Dies führte in der Browserumsetzung dazu, dass gegenüber <pre> oder dem vergleichbaren (ebenfalls abgeschafften) <XMP> oft ein kleinerer Schrifttyp verwendet wurde.

Das Element wird in aktuellen Browsern weiterhin unterstützt, sollte aber nicht länger eingesetzt werden. Verwenden Sie stattdessen <pre> (siehe Abschnitt 14.66, *pre*).

Beispiel: **[zB]**

```
<listing>
&lt;script type="text/vbs"&gt;
' eine Variablendeklaration mit dem Schl&uuml;sselwort "Dim"
Dim der_text
' eine Wertzuweisung
der_text = "Hallo Welt! Dies ist Visual Basic Script!"
' ein Funktionsaufruf - hier eine "Message-Box"
msgbox(der_text)
&lt;/script&gt;
</listing>
```

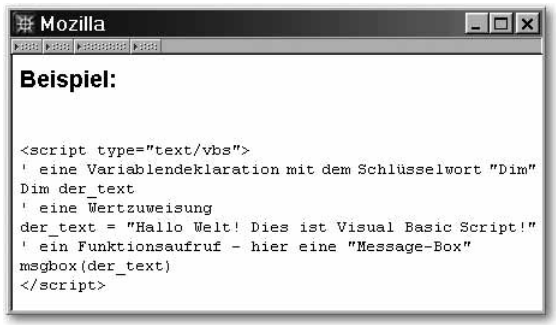

Abbildung 14.60 Ein mit dem <listing>-Container präsentierter Quellcodeblock

14.52 map

Der Container `<map>` beinhaltet in Form einer Reihe von `<area>`-Elementen die Definitionen einer clientseitigen Imagemap.

14.52.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Container, unsichtbar
XHTML-Modul	Client-side Image Map
Kontext	An beliebigen Stellen im `<body>` des Dokuments
Erlaubte Inhalte	Beliebige Anzahl von `<area/>`-Elementen
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01, XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile *nicht enthalten* in XHTML mobile profile iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<map> ... </map>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
map  {display:none; }
```

14.52.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `id`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–

Internet Explorer Events	onafterupdate, onbeforecut, onbeforepaste, onbeforeupdate, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onerrorupdate, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onrowenter, onrowexit, onscroll, onselectstart

Elementspezifische Attribute

W3C	**name**

14.52.3 Offizielle Attribute – W3C

name

Wert	*nmtoken*
Typ	Ein Bezeichner in Form eines NMTOKEN
Verwendung	**Obligatorisch**

Das obligatorische name-Attribut nennt einen Bezeichner des <map>-Containers, über den dieser von einem Objekt oder Bild aus, das als Imagemap dient, per Fragment-Identifier mittels dessen usemap-Attributs angesprochen wird.

```
<map name="map_bezeichner"> ... </map>
```

wird angesprochen über (im Falle eines Image-Objekts)

```
<img src="..." usemap="#map_bezeichner" />
```

14.52.4 Erläuterung zum Element

Das <map>-Element dient als unsichtbarer Container für <area>-Definitionen, die eine clientseitige Imagemap definieren. Gleichzeitig stellt der <map>-Container über sein name-Attribut die IDREF-Beziehung zum usemap-Attribut desjenigen Objekts her, das als Imagemap dienen soll. Hierbei handelt es sich meist um eine mit <img> eingebundene Grafik. Ebenso gut kann jedoch ein mit <object> eingebundenes beliebiges Medienobjekt oder ein über <applet> eingebundenes Java-Applet eine Map-Definition zugewiesen erhalten.

Barrierefreiheit

Imagemaps sollen immer clientseitig eingerichtet werden, da sie nur dann mit der Tastatur (d.h. auch ohne Maus) angesteuert werden können (siehe WAI 9.1).

14.53 marquee

Proprietäres Element: Das `<marquee>`-Element stellt einen Textstring innerhalb seines Inhaltsbereiches als Lauftext dar.

14.53.1 Elementeigenschaften

Einführung	Microsoft (IE2)
Elementtyp	Inlinecontainer (IE), Blockcontainer (sonst)
XHTML-Modul	– (non-standard)
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	– (nicht spezifiziert)
Mobil-Spezifikationen	iMode HTML 2.0, 3.0, 4.0, 5.0

Syntaxbeispiel:

```
<marquee> ... </marquee>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

–

14.53.2 Attribute (Übersicht)

Standardattribute

HTML	inoffiziell: `class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	inoffiziell: `class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–

Internet Explorer Events	onbeforecopy, onbeforecut, onbeforeeditfocus, onbeforepaste, **onbounce**[37], oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onerrorupdate, **onfinish**[37], onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, **onstart**[37], onselectstart

Elementspezifische Attribute

W3C	–
Internet Explorer	behavior, bgcolor, direction, height, hspace, loop, scrollamount, scrolldelay, truespeed, vspace, width *Access-, Databinding-, Edit-Attribute*

14.53.3 Proprietäre Attribute – Internet Explorer

behavior

Wert	*"scroll"* | *"slide"* | *"alternate"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das behavior-Attribut bestimmt die Art der Bewegung der Zeichenkette innerhalb des <marquee>-Bereiches.

- behavior="scroll" (Default)
  bewirkt die wiederholte Bewegung der Zeichenkette in die Richtung, die mittels des direction-Attributs festgelegt wird. Die Defaultrichtung ist von rechts nach links.
- behavior="slide"
  bewirkt das Anhalten des Marquee-Textes, sobald das letzte Zeichen der Zeichenkette innerhalb des Marquee-Bereiches erschienen ist.
- behavior="alternate"
  bewirkt das Wechseln der Bewegungsrichtung nach jedem vollständigen Durchlaufs des Marquee-Textes.

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

37 Spezielle Eventhandler für <marquee>.

Das Attribut `bgcolor` bestimmt die Hintergrundfarbe des Inhaltsbereichs des `<marquee>`-Elements. In Abwesenheit des Attributs gilt die Hintergrundfarbe des Dokuments (d.h. der Bereich ist transparent).

```
<marquee bgcolor="#ffff00">Hintergrund gelb</marquee>
```

direction

Wert	*"left"* | *"right"* | *"up"* | *"down"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `direction`-Attribut legt die Laufrichtung des Marquee-Effekts fest. Möglich sind die Angaben `"right"` (Laufrichtung links nach rechts), `"left"` (Laufrichtung rechts nach links), `"up"` (Zeilenbewegung von unten nach oben) und `"down"` (Zeilenbewegung von oben nach unten). Defaultrichtung ist `"left"`.

```
<marquee direction="right">Text läuft nach rechts</marquee>
```

height

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `height`-Attribut bestimmt die Höhe des Inhaltsbereiches des `<marquee>`-Elements in Pixel oder Prozent der Höhe des Browserfensters bzw. des unmittelbar hierarchisch übergeordneten Elements (Body, Frame, Tabellenzelle etc.).

hspace

Wert	*integerwert*
Typ	Ganzzahl als Pixelwert
Verwendung	Optional

Das `hspace`-Attribut nennt einen Pixelwert (Ganzzahl) für den symmetrisch links und rechts vom Marquee-Element freizuhaltenden Raum.

loop

Wert	–1 | number
Typ	Ganzzahl als Zahl der Widerholungen oder –1
Verwendung	Optional

Das `loop`-Attribut bestimmt mit einer positiven Ganzzahl größer gleich Null die Anzahl der Durchläufe des Lauftexts. Der Wert `"0"` unterbindet die Wiederholung. Der Defaultwert `"-1"` bewirkt eine endlose Wiederholung.

scrollamount

Wert	*number*
Typ	Ganzzahl
Verwendung	Optional

Das `scrollamount`-Attribut bestimmt indirekt die Geschwindigkeit des Lauftextes, indem es die Pixelabstände zwischen den einzelnen Scrollzuständen des `<marquee>`-Effekts festlegt. Der Zeitabstand zwischen den Bewegungsschritten wird durch das `scrolldelay`-Attribut und das `truespeed`-Attribut festgelegt.

scrolldelay

Wert	*number*
Typ	Ganzzahl
Verwendung	Optional

Das `scrolldelay`-Attribut bestimmt den Zeitabstand in Millisekunden, der zwischen zwei Scrollzuständen des `<marquee>`-Effekts liegt. Wird kein `scrolldelay`-Attribut gesetzt, so liegt der Defaultwert bei 60 ms, was dem Standardzeitraster für Marquee-Bewegungen entspricht.

Hinweis: Damit Zeitraster unter 60 Millisekunden möglich werden, ist das gleichzeitige Setzen des Attributs `truespeed="true"` erforderlich. Ansonsten wird der `scrolldelay`-Wert auf Vielfache von 60 ms gerundet. Werte unter 60 ms werden aufgerundet, darüber wird entsprechend auf Vielfache (120 ms, 180 ms usw.) auf- oder abgerundet. **[«]**

truespeed

Wert	*"true"* | *"false"*
Typ	Festgelegte boolesche Bezeichner
Verwendung	Optional

Das `truespeed`-Attribut bestimmt die Auswertung des `scrolldelay`-Attributs und damit die exakte Zeitsteuerung der Bewegung des Marquee-Lauftextes.

- `truespeed="false"` (Default)
  Der übergebene `scrolldelay`-Wert wird entsprechend dem 60 Millisekunden Zeitraster gerundet, wobei Werte von 59 ms und darunter auf 60 ms aufgerundet werden. Für kleine `scrolldelay`-Werte ist die tatsächliche Bewegungsgeschwindigkeit dementsprechend zu langsam:

  ```
  scrolldelay="6" scrollamount="10" truespeed="false"
  ```

  Das Zeitraster wird von 6 auf 60 Millisekunden aufgerundet. Der Marquee-Text bewegt sich daher alle 60 ms um 10 Pixel, also um Faktor 10 langsamer als beabsichtigt.
- `truespeed="true"`
  Es wird exakt das übergebene `scrolldelay`-Intervall als Zeitraster für die Marquee-Bewegung verwendet.

  ```
  scrolldelay="6" scrollamount="10" truespeed="true"
  ```

  Das Zeitraster wird durch das `truespeed`-Attribut auf 6 ms verkürzt. Der Marquee-Text bewegt sich wie beabsichtigt alle 6 ms um 10 Pixel, also um Faktor 10 schneller als ohne gesetztes `truespeed`-Attribut.

vspace

Wert	*number*
Typ	Ganzzahl als Pixelwert
Verwendung	Optional

Das `vspace`-Attribut nennt einen Pixelwert (Ganzzahl) für den über und unter dem Marquee-Bereich freizuhaltenden Raum.

width

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `width`-Attribut bestimmt die Breite des Inhaltsbereiches des `<marquee>`-Elements in Pixel oder Prozent der Breite des Browserfensters bzw. des unmittelbar hierarchisch übergeordneten Elements (Body, Frame, Tabellenzelle etc.).

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute
`datafld = "feldname"` (IE 4)

`dataformatas = "text" | "localized-text" | "HTML"`

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute
`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.53.4 Erläuterung zum Element

Das `<marquee>`-Element wurde von Microsoft für den Internet Explorer 2.0 eingeführt und dient zum Erzeugen eines Lauftextes. Das Element verhält sich in Nicht-Microsoft-Browsern wie ein Blockcontainer. Es bewirkt also stets einen Zeilenumbruch vor und nach dem Elementbereich, auch wenn es auf eine geringere Breite skaliert wird, als der übergeordnete Block zur Verfügung stellt.

Im Internet Explorer hingegen ist `<marquee>` prinzipiell ein Inlineelement, das sich jedoch ebenfalls wie ein Blockcontainer verhält, solange es nicht auf eine festgelegte Breite gesetzt ist. Da die Defaultbreite stets 100 % beträgt, verdrängt es benachbarten Inlinecontent und verursacht wiederum Zeilenumbrüche.

Zur Einbindung reicht Folgendes aus:

```
<marquee>
 Dieser Text l&auml;uft von rechts nach links.
</marquee>
```

Der Text im Containerinneren scrollt in ständiger Wiederholung von rechts nach links durch den Marquee-Bereich. Mit dem `behavior`-Attribut lassen Sie ihn nach Durchqueren des Darstellungsbereiches die Laufrichtung wechseln:

```
<marquee behavior="alternate">
 Ein Text mit wechselnder Laufrichtung.
</marquee>
```

Ein weiterer Wert, `"slide"`, stoppt den Text, sobald er vollständig im Inneren des Darstellungsbereichs angekommen ist. Mit `height` und `width` können Sie die Größe des Marquee-Bereiches festlegen (beachten Sie, dass das Verhalten der Browser gegenüber umliegenden Inlineelementen stark schwankt).

```
<marquee width="200" height="20">
 Ein Newsticker mit festgelegtem Darstellungsbereich.
</marquee>
```

Innerhalb des Lauftextes können auch Inlineelemente platziert werden, beispielsweise Hyperlinks. Sie sollten in Betracht ziehen, in diesem Fall `behavior` auf `"slide"` zu setzen, um die »Trefferquote« des Users zu verbessern:

```
<marquee width="200">
 Ein Newsticker mit <a href="meldung.html">Hyperlink</a>.
</marquee>
```

Ebenso kann eine Grafik (beispielsweise ein animiertes GIF) innerhalb des Marquee-Textes erscheinen, oder diesen komplett ersetzen. Mittels `scrollamount` und `scrolldelay` lässt sich das Zeitverhalten (Geschwindigkeit) des Marquee-Effekts steuern:

```
<marquee scrollamount="5" scrolldelay="5">
 <img src="hoppelhase.gif" alt="Hase hoppelt durchs Bild">
</marquee>
```

Unterstützung durch Browser

Der `<marquee>`-Container wird den meisten aktuellen Browsern zumindest rudimentär unterstützt, ist jedoch Teil keiner offiziellen Spezifikation (auch die Mobil-Spezifikationen, die `<marquee>` enthalten, sind keine Standards).

Die Implementierungen in Nicht-Microsoft-Browsern variieren jedoch stark und umfassen in keinem Fall die speziellen Eventhandler `onbounce`, `onfinish` und `onstart` zur Steuerung von Scripten.

Der Eventhandler `onbounce` feuert, wenn der Lauftext die Grenze des Marquee-Bereichs erreicht hat und die Laufrichtung wechselt (hierfür muss `behaviour="alternate"` gesetzt sein):

```
<!-- Alertbox bei Erreichen des Marque-Randes (nur IE): -->
<marquee behavior="alternate" loop="5"
 onbounce="alert('Text hat Rand erreicht!')">
 Dieser Lauftext triggert ein JavaScript!
</marquee>
```

Weitere IE-Eventhandler für `<marquee>` sind:

- `onfinish`
  feuert beim Beenden des Schleifendurchlaufs.
- `onstart`
  feuert zu Beginn des Schleifendurchlaufs.

Barrierefreiheit und <marquee>

In Bezug auf die Barrierefreiheit ist das <marquee>-Element weniger deswegen problematisch, dass nicht alle User Agents das Element unterstützen – in diesem Fall würde immerhin der Elementinhalt gezeigt. Schwerwiegender ist die Tatsache, dass Laufgeschwindigkeit und weitere Parameter in User Agents, die das Element unterstützen, nicht gesteuert werden können. So kann die Geschwindigkeit im Bedarfsfall nicht herabgesetzt werden; auch ist nicht in jedem Falle der gesamte Textinhalt auf einen Blick sichtbar (siehe BITV 7.3.1).

Zukunft des <marquee>-Elements

Große Chancen auf eine Aufnahme in die offiziellen Spezifikationen des W3C bestehen für den <marquee>-Tag derzeit nicht. Anders sieht dies eventuell beim Konkurrenzvorschlag »Web Application« 1.0 der WHAT WG aus, in der der <marquee>-Container zumindest als potentiell erlaubter Inhalt anderer Elemente namentlich erwähnt wird. Zu einer expliziten Spezifikation des Elements schweigt sich der Vorschlag hingegen aus.

14.54 menu

Deprecated: Das <menu>-Element dient als Container für Listenelemente <li>, die ein kompaktes Navigationsmenü darstellen sollen.

14.54.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	Legacy
Kontext	Unmittelbar in <body> sowie in <applet>, <blockquote>, <button>, , <dd>, <del>, , <fieldset>, <form>, <iframe>, <ins>, , <map>, <noframes>, <noscript>, <object>, <td>, <th>
Erlaubte Inhalte	Elemente vom Typ
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0/4.01 (tr/fs) XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0/4.01 strict, XHTML 1.1, 2.0
Mobil-Spezifikationen	cHTML *nicht enthalten* in HTML 4.0 mobile, XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<menu> ... </menu>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
menu    { margin-left: 40px }
```

14.54.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`compact`
Internet Explorer	`type` *Access-, Edit-Attribute*
Netscape	`type`
Web Application 1.0	`autosubmit`, `label`, `type`[38]

14.54.3 Offizielle Attribute – W3C

compact

Wert	*"compact"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

38 Andere Bedeutung als bei IE und Netscape.

Deprecated (Präsentationsattribut): Das `compact`-Attribut reduziert den Abstand zwischen den Listenelementen der Liste und soll eine kompaktere Darstellung der Liste bewirken. In der Praxis wird das Attribut von kaum einem Browser unterstützt. Ohnehin sollte eine `<menu>`-Liste bereits kompakter dargestellt werden als eine normale `<ul>`-Liste.

14.54.4 Proprietäre Attribute – Internet Explorer:

type

Wert	*"disc"* | *"circle"* | *"square"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `type`-Attribut bestimmt im Internet Explorer die Art des Bullets der `<menu>`-Liste. Der übliche Defaultwert ist `"disc"`.

Der Wert des `type`-Attributs eines enthaltenen Listenelements `<li>` hat stets Vorrang vor der defaultmäßig oder per `type`-Attribut vorgegebenen Bulletform des umgebenden Listencontainers.

- `type="disc"`
  erzeugt ein Bullet in Form einer schwarzen Scheibe.
- `type="circle"`
  erzeugt ein Bullet in Form eines Kreises.
- `type="square"`
  erzeugt ein Bullet in Form eines schwarzen Quadrats.

Anmerkung: Die Spezifikation sieht eigentlich eine Darstellung von Menülisten ohne Bullets vor.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.54.5 Proprietäre Attribute – Netscape

type

Wert	*"disc"* | *"circle"* | *"square"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `type`-Attribut bestimmt im Netscape Navigator die Art des Bullets der `<menu>`-Liste. Die übliche Defaultdarstellung ist `"disc"`. Das Attribut entspricht der Implementierung beim Internet Explorer.

14.54.6 Erläuterung zum Element

Das `<menu>`-Element zählt zu den vom W3C als »missbilligt« eingestuften Elementen und sollte nicht länger eingesetzt werden. Es ist allerdings weiterhin Teil der XHTML 1.0 *transitional*- und *frameset*-DTDs und auch Teil des Legacy-Moduls von XHTML.

Die Idee hinter dem Container `<menu>` als eigenständige Listenform bestand darin, dass eine Navigationsliste gewöhnlich aus kurzen, einzeiligen Einträgen bestehen würde, die demzufolge kompakter dargestellt werden könnte. Auch Bullets vor den Listenpunkten waren ursprünglich nicht vorgesehen.

In der Browserpraxis wurde allerdings nirgends eine von gewöhnlichen `<ul>`-Listen abweichende Darstellung implementiert (analog zur ebenfalls als abgeschafft geltenden Directory-Liste `<dir>`). Das W3C schlägt vor, an Stelle von `<menu>` stets die »unordered list« `<ul>` einzusetzen.

[zB] *Beispiel:*

```
<menu>
<li>Home</li>
<li>&Uuml;ber uns</li>
<li>Kontakt</li>
</menu>
```

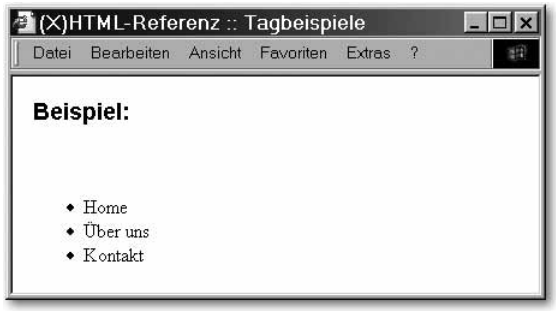

Abbildung 14.61 Eigentlich falsch dargestellte Menüliste in Mozilla

Anmerkung: In der durch die WHAT WG propagierten Spezifikation zu »Web Applications 1.0« (populär als »HTML 5« bezeichnet) wird vorgeschlagen, das `<menu>`-Element *beizubehalten* und sogar funktional zu erweitern[39].

Die Generierung von »echten« Navigationsmenüs wird dort durch die neuen Attribute `type` (nicht für Bullets, sondern zur Wahl zwischen `"popup"`- und `"toolbar"`-Menüs), `label` (Marker für verschachtelte Menüs) und `autosubmit` (für unmittelbare Aktion) unterstützt. In XHTML 2.0 wird hingegen ein konzeptionell vergleichbarer, neuer Listencontainer `<nl>` propagiert[40].

14.55 meta

Das `<meta>`-Element fügt dem Dokument unsichtbare Meta-Informationen wie Beschreibung, Schlagworte, Autor oder Zeichenkodierung bei.

14.55.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Head
XHTML-Modul	Metainformation
Kontext	Im `<head>`-Container
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01, XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

39 Siehe: http://whatwg.org/specs/web-apps/current-work/#the-menu.
40 Siehe: http://www.w3.org/TR/xhtml2/mod-list.html#edef_list_nl.

Syntaxbeispiel:

```
<meta />
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
/* display-Property geerbt von head (inheritance): */
meta   { display: none; }
```

14.55.2 Attribute (Übersicht)

Standardattribute

HTML	`lang`
XHTML	`dir`, `xml:lang`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

W3C	**`content`**, `http-equiv`, `name`, `scheme`

14.55.3 Offizielle Attribute – W3C

content

Wert	*cdata*
Typ	eine beliebige Zeichenkette (CDATA)
Verwendung	Obligatorisch

Das `content`-Attribut enthält die eigentliche Information des `<meta>`-Elements, dessen Art wahlweise durch das `name`- oder das `http-equiv`-Attribut bestimmt wird. Je nach Art der Information ist die Struktur des Attributwertes mehr oder weniger reglementiert.

Das Attribut ist *case-sensitive*.

Hinweis: Die Inhalte des `content`-Attributs werden nach den normalen Regeln geparst. Es ist daher möglich und sinnvoll, Sonder- und Steuerzeichen durch HTML-Entities oder Character-Entities zu ersetzen. **[«]**

http-equiv

Wert	*cdata*
Typ	http-equiv-Bezeichner
Verwendung	Optional

Das Attribut `http-equiv` beinhaltet Anweisungen an den Webserver bzw. aus dem HTTP-Protokoll abgeleitete Begriffe. Der Server kann Inhalte der `http-equiv`-Meta-Elemente vor Ausliefern des Dokuments auslesen und dem HTTP-Header beifügen. Geschieht dies nicht, so ist der Browser angehalten, die Angaben so zu interpretieren, als ob sie im HTTP-Header stehen würden.

Übliche Werte sind: `"content-type"`, `"content-style-type"`, `"content-script-type"`, `"expires"`, `"refresh"`, `"reply-to"`

Das Attribut ist nicht *case-sensitive*.

Hinweis: Konflikte zwischen Angaben aus dem HTTP-Header des Servers und `http-equiv`-Angaben sollten vermieden werden – dies betrifft beispielsweise die Angabe des Content-Typs. **[«]**

name

Wert	*cdata*
Typ	name-Bezeichner
Verwendung	Optional

Das `name`-Attribut beinhaltet Informationen für den Client, der die Seite aufruft. Hierbei handelt es sich im Allgemeinen um den Webbrowser, aber auch um Suchmaschinen-Robots und vergleichbare Clients.

Das Attribut ist *case-sensitive*.

Übliche Werte sind: `"abstract"`, `"audience"`, `"author"`, `"copyright"`, `"date"`, `"description"`, `"generator"`, `"keywords"`, `"publisher"`, `"revisit-after"`, `"robots"` sowie eine große Anzahl von Dublin Core-Typen (DC.).

Hinweis: Es besteht keine Verpflichtung für einen Client, einen bestimmten Wert des `name`-Attributs zu unterstützen. Entsprechend werden nicht erkannte Meta- **[«]**

Elemente ignoriert. Auf der anderen Seite besteht die Möglichkeit, eigene `name`-Werte einführen, die von entsprechend programmierten Clients (beispielsweise innerhalb eines Intranets) genützt werden können.

scheme

Wert	*cdata*
Typ	Bezeichner für Metaprofil
Verwendung	Optional

Das `scheme`-Attribut verweist auf eine Formatdefinition für den `content`-Inhalt des Meta-Elements. Dies steht meist in Verbindung mit einem Meta-Profil, in dem sich die eigentlichen Formatdefinitionen befinden, das über das `profile`-Attribut im `<head>`-Element mit dem Dokument verknüpft ist.

Das Attribut ist *case-sensitive*.

Hier wird die name-Angabe `"buch-id"` als ISBN-Nummer identifiziert:

```
<meta scheme="ISBN" name="buch-id" content="3-89842-361-1"/>
```

Praxistipp

Für **Meta-Daten nach Dublin Core** existieren eine Reihe registrierter DC-Schema-Typen, die in diesem Zusammenhang referenziert werden können:

DC Schemaname	Benennung und Anwendung
`Box`	Geographische Angaben
`DCMIType`	Dublin Core Metadata Initiative Type; allg. DC Typ
`DDC`	Dewey Decimal Classification; Klassifikationssystem für Bibliotheken
`IMT`	Internet Media Type; MIME-Type Bezeichner
`ISO3166`	Länderkennzeichen nach ISO3166
`ISO639-2`	Sprache nach ISO639–2
`LCC`	Library of Congress Class; Klassifikationssystem
`LCSH`	Library of Congress Subject Heading; Klassifikationssystem
`MESH`	Medical Subject Headings; Klassifikation im medizinischen Bereich
`NLM`	National Library of Medicine; Klassifikation im medizinischen Bereich
`Period`	Angabe eines Zeitraumes

DC Schemaname	Benennung und Anwendung
Point	Angabe eines geographischen Ortes
RFC1766	Sprache nach RFC1766
RFC3066	Sprache nach RFC3066
TGN	Thesaurus of Geographics Names; Ortsnamen
UDC	Universal Decimal Classification
URI	URI-String
W3CDTF	Zeitstempelwert gemäß W3C

Die Angabe des Gültigkeitszeitraumes eines Dokuments erfolgt mit `scheme="Period"`:

```
<meta name="DC.date.valid" scheme="Period"
 content="start=2006-07-15; end=2006-12-31; scheme=W3CDTF;
 name=Webeditoren aktuell"/>
```

Beachten Sie, dass DC-Schematypen nur im Zusammenhang mit DC-Nametypen sinnvoll sind!

14.55.4 Erläuterung zum Element

Mit Hilfe des `<meta>`-Elements können im Dokumentkopf Informationen über Inhalt und Bedeutung des Dokuments gemacht werden (Meta-Informationen), die es erleichtern, das Dokument zu klassifizieren und so seine Inhalte besser oder gezielter zugänglich zu machen.

Adressat von Meta-Informationen sind einerseits **Suchmaschinen**, die durch `<meta>`-Tags Hilfestellung beim Katalogisieren von Webseiten erhalten, andererseits spezialisierte **Intranet-Anwendungen**, die beispielsweise Bibliotheksverzeichnisse erstellen, in deren Rahmen Relationen zwischen Dokumenten oder deren Rolle beschrieben werden muss. Für diese Aufgabe sind besonders Meta-Angaben nach **Dublin Core** geeignet, die nachfolgend ausführlich beschrieben werden. Alle diese Angaben werden durch das `name`-Attribut mit bestimmten, vordefinierten Properties vorgenommen. Die übergebenen Werte (`content`-Attribut) können mittels der Angabe eines Schemas (`scheme`-Attribut) qualifiziert werden, womit bekräftigt wird, dass die Angaben einer bestimmten inhaltlichen Form folgen (als Datumswert, URI-String, Sprachangabe o.ä.).

Eine dritte Aufgabe des `<meta>`-Elements besteht in der Weitergabe von Angaben an den Webbrowser, die stellvertretend oder alternativ zu Informationen eingesetzt werden sollen, die ansonsten vom Webserver im Rahmen des HTTP-Hea-

ders übergeben werden. Solche Meta-Angaben erfolgen mit dem `http-equiv`-Attribut.

Die Bedeutung der Meta-Informationen in Bezug auf Suchmaschinen wird stark mystifiziert und ist weniger bedeutend als allgemein angenommen. Unter den nachfolgend beschriebenen `name`-Typen (die Liste ist nicht vollständig) befinden sich nur drei wirklich relevante, nämlich `description`, `keywords` und `robots`.

Dies soll Sie nicht davon abhalten, zusätzlich weitere Meta-Angaben mit `name` zu verwenden oder selbst eigene Typen einzuführen (es besteht in dieser Hinsicht kein Reglement) – allerdings müssen Sie auch für die auswertende Software selbst sorgen. Beachten Sie jedoch, dass überflüssige Meta-Angaben die Dateigröße der Webseiten aufblähen und das Verhältnis von eingesetztem HTML zur Informationsmenge ungünstig beeinflussen.

http-equiv-Typen

Meta-Elemente mit `http-equiv`-Attribut stellen das Äquivalent zu den gleichnamigen HTTP-Headerinformationen dar, die ein Webserver dem Dokument bei der Auslieferung mit auf den Weg gibt. Sie können daher als Erweiterung des tatsächlichen HTTP-Headers eingesetzt werden.

cache-control

Entspricht dem HTTP-Cache-Control-Mechanismus (nach RFC2068, Abschnitt 14.9). Hiermit kann dem Browser (unter anderem) untersagt werden, die Seite im Cache abzulegen.

```
<meta http-equiv="cache-control" content="no-cache"/>
```

content-language

Nennt die Sprache, in der der Textinhalt des Dokuments formuliert ist. Es dürfen auch Untertypen (Dialekte) einer Sprache genannt werden (beispielsweise `"en-US"` für amerikanisches Englisch).

```
<meta http-equiv="content-language" content="de"/>
```

Anmerkung: Das Meta-Element `content-language` bietet Suchmaschinen Hilfestellung beim Zuordnen des Inhalts zu einer Sprache.

content-script-type

Nennt explizit den MIME-Typ der für die Seite zum Einsatz kommenden Scriptsprache. Neben JavaScript kann auch VBScript, JScript oder eine andere Scriptsprache genannt werden.

```
<meta http-equiv="content-script-type" content="text/javascript"/>
```

Das Nennen der Scriptsprache hat keinen merklichen Einfluss auf die Interpretation per `<script>` eingebetteter Scripte durch den Browser. Maßgeblich ist jeweils der Wert des `type`-Attributs, dessen Einsatz für den `<script>`-Container ohnehin vorgeschrieben ist.

Anders ist dies im Falle so genannter *intrinsic scripts*, wie sie im Rahmen von Eventhandler-Attributen eingesetzt werden. Da hier keine direkte Nennung der Sprache möglich ist, interpretiert der Browser diese gemäß der Default-Scriptsprache (d.h. JavaScript), solange keine Content-Scriptsprache übergeben wird. Wollen Sie beispielsweise VBScript im Rahmen von Eventhandlern direkt verwenden, legen Sie VBScript als Content-Scriptsprache fest.

content-style-type

Nennt explizit den MIME-Typ der für die Seite eingesetzten Stylesheet-Sprache. Normalerweise wird `"text/css"` übergeben.

```
<meta http-equiv="content-style-type" content="text/css"/>
```

Anmerkung: Der `content-style-type` greift im Grunde nur für die Werte des `style`-Attributs (Inlinestyles), da in allen anderen Fällen der Contenttyp genannt werden kann. Sofern Sie das `style`-Attribut einsetzen (hiervon ist eher abzuraten), verwenden Sie vorsichtshalber auch dieses Meta-Element.

content-type

Gibt für den Browser den MIME-Typ und in der Seite verwendeten Zeichensatz an, wie er alternativ auch vom Server im Rahmen des HTTP-Headers übermittelt wird. Wenn Sie im Dokument den westeuropäischen Zeichensatz einsetzen, geben Sie als Content den Wert `"charset=ISO-8859-1"` an.

```
<meta http-equiv="content-type"
 content="text/html; charset=ISO-8859-1"/>
```

expires

Nennt ein Gültigkeitsdatum bis zu dessen Verstreichen das Dokument in einer gecacheten Version verwendet werden darf. Das Datum soll laut Reglement in der in RFC850 festgelegten Form übergeben werden.

```
<meta http-equiv="expires"
 content="Sun, 31 Dec 2006 00:00:00 GMT"/>
```

pragma

Untersagt die Speicherung der Seite auf einem Proxy-Server. Als Content muss `"no-cache"` übergeben werden.

```
<meta http-equiv="pragma" content="no-cache"/>
```

refresh

Gibt dem Browser die Anweisung, nach einer festgelegten Zeit entweder die aktuelle Seite neu zu laden oder, falls ein URL (absolut oder relativ) übergeben wird, zur angegebenen Adresse zu springen. Wird als Wartezeit 0 angegeben, erfolgt die Weiterleitung sofort.

```
<meta http-equiv="refresh" content="0;
URL=http://www.example.com/start.html"/>
```

Achtung – Problem mit Barrierefreiheit

Automatische Weiterleitung und Aktualisierung sind laut BITV 10.1 mit »den gängigen Hilfsmitteln nicht kompatibel«. Bis eine browserseitige Kontrolle der Funktionen implementiert ist, sollte nach WAI 7.4/7.5 und BITV 7.4/7.5 der `refresh`-Befehl nicht verwendet werden. (Serverseitig gesteuerte Weiterleitungen gelten hingegen als unproblematisch.) Es existieren jedoch Screenreader, die ein Meta-Refresh unterbinden können (z. B. HAL, **www.yourdolphin.com**).

set-cookie

Setzt einen Cookie mit dem übergebenen Inhalt, wobei gleichzeitig Gültigkeitsbegrenzung und Pfadangabe vorgenommen werden kann.

```
<meta http-equiv="set-cookie"
content="cookievalue=abcdefg;
expires=Friday, 29-Dec-06 20:00:00 GMT; path=/">
```

Anmerkung: Funktioniert nur mit (älteren) Netscape Browsern. Verwenden Sie stattdessen besser JavaScript.

name-Typen

Alle Meta-Angaben, die nicht Äquivalente zu HTTP-Headerinstruktionen darstellen, werden mit dem `name`-Attribut vorgenommen.

abstract

Enthält dem `<title>`-Container entsprechend eine kurze Beschreibung des Inhaltes der Ressource.

```
<meta name="abstract" content="Einf&uuml;hrung Webeditoren"/>
```

Anmerkung: Ursprünglich von IBM eingeführt. Die Relevanz für Suchmaschinen ist gering.

author

Autor des Inhalts der Ressource bzw. der für den Inhalt Verantwortliche.

```
<meta name="author" content="Curt Coder"/>
```

Anmerkung: Nicht-qualifizierte Autorenangabe geringer Relevanz. Stammt ursprünglich von Authoring Tools wie Netscape Gold. Für eine qualifizierte Autorenangabe können Sie den Dublin-Core-Typ `DC.creator` verwenden.

audience

Beschreibung der Zielgruppe der Ressource mit Stichworten, wobei mehrere, durch Komma getrennte Bezeichner übergeben werden können.

```
<meta name="audience"
 content="Webdesigner, Webprogrammierer, Programmierer"/>
```

Anmerkung: Richtet sich an die Suchmaschine **Fireball** (**www.fireball.de/**).

copyright

Der Rechteinhaber der Urheberrechte der Inhalte (meist der Verfasser).

```
<meta name="copyright" content="Curt Coder"/>
```

Anmerkung: Nicht-qualifizierte Copyright-Angabe geringer Relevanz. Stammt ursprünglich von Authoring Tools wie Netscape Gold. Für eine qualifizierte Rechteangabe können Sie den Dublin-Core-Typ `DC.rights` verwenden.

date

Das Datum der Veröffentlichung der Ressource.

```
<meta name="date" content="2006-07-15"/>
```

description

Ein kurzer Beschreibungstext zum Inhalt der aktuellen Ressource.

```
<meta name="description" content="&Uuml;bersicht und Vergleich
aktueller Webeditoren, Darstellung der Funktionen, Screenshots und
Bezugsquellen "/>
```

Hinweis: Überschreiten Sie möglichst nicht die Grenze von 150 bis 180 Zeichen (Leerzeichen eingerechnet) – Suchmaschinen berücksichtigen selten mehr. In der Ergebnisliste der Suchmaschinen dient die Meta-Beschreibung als Inhaltsangabe zur gefundenen Seite und sollte entsprechend sorgfältig formuliert werden. **[«]**

generator

Dieses Meta-Element wird von verschiedenen HTML-Editoren eingefügt.

```
<meta name="generator" content="Microsoft Frontpage"/>
```

[»] *Hinweis:* Das Element dient lediglich der Messung der Marktanteile der Editoren. Sie brauchen es von sich aus nicht einzusetzen und können es, wo vorhanden, gefahrlos entfernen. Für Suchmaschinen ist das Property irrelevant.

keywords

Eine Liste kommagetrennter Schlüsselwörter, die Suchmaschinen zum Verschlagworten des Dokuments verwenden sollen.

```
<meta name="keywords" content="Editor, HTML-Editor"/>
```

[»] *Hinweis:* Setzen Sie die wichtigsten Keywords an den Anfang, da Suchmaschinen nicht beliebig lange Keywortlisten einlesen.

page-type

Beschreibt den inhaltlichen Typ der Webseite, wobei mehrere Kategorien durch Komma getrennt übergeben werden können.

```
<meta name="page-type" content="FAQ"/>
```

Anmerkung: Richtet sich an die Suchmaschine **Fireball** (**www.fireball.de/**) und daher für den allgemeinen Gebrauch wenig sinnvoll. Ob dieses Meta-Element aktuell noch unterstützt wird, ist unbekannt.

page-topic

Beschreibt in Stichworten das Thema der Seite, wobei mehrere Stichwörter durch Komma getrennt werden.

```
<meta name="page-topic" content="Software" />
```

Anmerkung: Richtet sich an die Suchmaschine **Fireball** (**www.fireball.de/**). Das Property entspricht ungefähr dem `keywords`-Property.

publisher

Enthält den Namen des für die Inhalte verantwortlichen Autors.

```
<meta name="publisher" content="Curt Coder" />
```

Anmerkung: Geringe bis keine Relevanz für Suchmaschinen. Stammt ursprünglich aus dem HTML-Export von MS-Word.

revisit-after

Dient (in der Theorie) dazu, einen Crawler zu veranlassen, die Seite nach dem angegebenen Zeitraum erneut zu besuchen.

```
<meta name="revisit-after" content="7 days" />
```

Anmerkung: Das Meta-Property `revisit-after` wird von so gut wie keiner Suchmaschine unterstützt und ist somit für den praktischen Einsatz irrelevant.

robots

Reglementiert das Verhalten des Suchmaschinen-Crawlers in Bezug auf die aktuelle Seite, wobei das Indizieren der Inhalte und das Verfolgen von Links, die von der aktuellen Seite ausgehen, gestattet oder verboten werden kann.

Diese Angabe verbietet das Indizieren der Seiteninhalte:

```
<meta name="robots" content="noindex" />
```

Achtung: Die `robots`-Angabe dient der *Beschränkung* der Suchmaschinenaktivität. Lassen Sie diese Meta-Angabe weg, geht der Suchmaschinen-Crawler davon aus, dass sowohl Indizieren als auch Linkverfolgung gestattet sind! **[!]**

Sie können als Content entweder die Schlüsselwörter `none` und `all` oder die Schlüsselwörter `index`, `noindex`, `follow`, `nofollow` sowie gegebenenfalls `noarchive` und `nosnippet` in beliebigen Kombinationen einsetzen (Trennung mit Komma):

- `content="index, nofollow"`
  Inhalte indizieren, Links nicht verfolgen.
- `content="noindex, follow"`
  Inhalte nicht indizieren, Links verfolgen.
- `content="noindex, nofollow"` oder `content="none"`
  Inhalte nicht indizieren, Links nicht folgen.
- `content="index, follow"` oder `content="all"` (Default)
  Inhalte indizieren, Links verfolgen. Diese Angabe können Sie auch unterlassen, da sie dem Standard-Crawlerverhalten entsprechen.
- `content="noarchive, nosnippet"`
  Seite nicht archivieren und auch keine Seitenauszüge (»snippets«) ausgeben, die Suchworte enthalten (diese Angabe zielt speziell auf Google und hat für andere Suchmaschinen keine Relevanz).

Anmerkung: Anstelle des pauschalen Meta-Properties `robots` können Sie auch gezielt einzelne Suchmaschinen-Crawler adressieren. So hört Google auf den Bezeichner `googlebot`, der Crawler von MSN auf `msnbot`:

```
<meta name="googlebot" content="noarchive, nofollow" />
```

Google unterstützt `noindex`, `nofollow`, `noarchive` und `nosnippet`.

```
<meta name="msnbot" content="index, nofollow" />
```

MSN unterstützt `noindex` und `nofollow`.

Praxistipp zu name-Properties

Im Hinblick auf die Zugänglichkeit für Suchmaschinen sollten Sie auf die Verwendung der Meta-Angaben `description` und `keywords` nicht verzichten. Die eigentlichen Inhalte der Seite werden jedoch stets höher bewertet als etwaige Schlüsselwörter. Interessant ist die `robots`-Angabe, sofern Sie das Verhalten von Suchmaschinen-Crawlern seitenbezogen reglementieren wollen.

Dublin-Core name-Typen

Initiiert durch die »Dublin Core Metadata Initiative«[41] (DCMI) wurde ein Meta-Datenschema zur Beschreibung von Dokumenten und anderen Objekten im Internet erstellt, kurz selbst als »Dublin Core« bezeichnet. Die Meta-Daten werden über das `name`-Attribut in das `<meta>`-Element eingebunden, wobei dem Bezeichner zur Kennzeichnung das Kürzel DC vorangestellt wird.

Der **Dublin Core Metadata Element Set, Version 1.1** unterscheidet 15 Haupttypen (»elements«) von Meta-Daten, denen der »recommended«-Status zuerkannt ist. Weitere Eintragungen existieren mit Status »conforming«, »endorsed«, »registered« und »obsolete«. Es ist jedoch seit 1997 Konsens, die »recommended«-Liste nicht zu erweitern.

`DC.contributor`	`DC.format`	`DC.rights`
`DC.coverage`	`DC.identifier`	`DC.source`
`DC.creator`	`DC.language`	`DC.subject`
`DC.date`	`DC.publisher`	`DC.title`
`DC.description`	`DC.relation`	`DC.type`

Tabelle 14.2 Elemente der Meta-Daten nach Dublin Core

41 Dublin Core Metadata Initiative: http://dublincore.org; Eine aktuelle Übersicht über die DC-Elemente erhalten Sie unter http://dublincore.org/documents/dcmi-terms/.

Zu einigen der Elemente existieren Untertypen, so genannte »refinements«, deren Bezeichner mittels Punktsyntax an den des Haupttyps angehängt werden.

Beispiel: **[zB]**

`DC.coverage` mit »refinement« `spatial` ergibt: `DC.coverage.spatial`

Es werden 28 Typen von »refinements« unterschieden (auch »subfields« genannt), denen der »recommended«-Status zuerkannt wird:

Element	Refinement
DC.coverage	spatial, temporal
DC.date	created, valid, available, issued, modified
DC.description	tableOfContents, abstract, extent, medium
DC.format	extent, medium
DC.relation	conformsTo, isVersionOf, hasVersion, isReplacedBy, replaces, isRequiredBy, requires, isPartOf, hasPart, isReferencedBy, references, isFormatOf, hasFormat
DC.title	alternative

Tabelle 14.3 Refinements der Meta-Elemente nach Dublin Core

Groß- und Kleinschreibung spielt für die Syntax keine Rolle. Zum Zweck der Lesbarkeit existiert die Konvention, den Namen der DC-Elemente klein und den vorausgehenden Identifier groß zu schreiben (z. B. `DC.creator`).

DC.contributor

Nennt Personen, Institutionen oder Organisationen, die zur Erstellung der Ressource Beiträge geliefert haben. Mehrere Einträge werden durch Komma getrennt.

```
<meta name="DC.contributor"
 content="Curt Coder, Torsten Texter, Webeditor INC"/>
```

DC.coverage

Nennt den räumlichen und/oder zeitlichen Bezugsrahmen der Ressource, wobei Orts- und Zeitangaben übergeben werden können. Stellt den Obertyp von `DC.coverage.spatial` und `DC.coverage.temporal` dar.

```
<meta name="DC.coverage" content="Europa, 2006"/>
```

DC.coverage.spatial

Nennt den räumlichen Bezugsrahmen der Inhalte der Ressource in Form von Orts-, Regions- und Ländernamen oder Längen- und Breitenangaben. `DC.coverage.spatial` ist ein Untertyp (»*refinement*«) von `DC.coverage`.

```
<meta name="DC.coverage.spatial" content="Europa"/>
```

DC.coverage.temporal

Nennt den zeitlichen Bezugsrahmen der Inhalte der Ressource, wobei ein Datum, Jahreszahlen sowie Perioden- und Epochennamen gestattet sind. `DC.coverage.temporal` ist ein Untertyp (»*refinement*«) von `DC.coverage`.

```
<meta name="DC.coverage.temporal" content="1999-2005"/>
```

DC.creator

Nennt die Person, Institution oder Organisation, die für die Erstellung der Ressource verantwortlich ist.

```
<meta name="DC.creator" content="Curt Coder"/>
```

Anmerkung: entspricht `name="author"`

DC.date

Nennt das Datum der Erstellung der Ressource in ihrer gegenwärtigen Form (kann `DC.date.modified` entsprechen).

```
<meta name="DC.date" content="2006-09-10"/>
```

DC.date.created

Nennt das Datum der (erstmaligen) Erstellung der Ressource als Kalenderdatum oder Zeitstempelwert.

```
<meta name="DC.date.created" content="2006-07-15"/>
```

```
<meta name="DC.date.created" content="2006-07-15T19:45:00"/>
```

DC.date.valid

Nennt das Datum oder den Zeitraum der Gültigkeit der Ressource.

```
<meta name="DC.date.valid" content="2006-07-15"/>
```

Für Zeiträume eignet sich der DC-Schemabezeichner »Period«, damit ein Start- und ein Endtermin genannt werden kann.

```
<meta name="DC.date.valid" scheme="Period"
 content="start=2006-07-15; end=2006-12-31; scheme=W3CDTF;
 name=Webeditoren aktuell"/>
```

Anmerkung: Innerhalb der Meta-Daten des obigen Beispiels wird Bezug auf den DC-Schematyp W3CDTF genommen (`scheme=W3CDTF;`), was aussagt, dass es sich hier um Zeitstempelwerte gemäß »W3C Date Time Format« handelt.

DC.date.available

Nennt das Datum oder den Zeitraum der Verfügbarkeit der Ressource.

```
<meta name="DC.date.available" content="2006-07-15"/>
```

Hier kann ebenfalls der DC-Schemabezeichner »Period« eingesetzt werden, damit ein Start- und ein Endtermin genannt werden kann.

```
<meta name="DC.date.available" scheme="Period"
 content="start=2006-07-15; end=2006-12-31; scheme=W3CDTF;
 name=Webeditoren aktuell"/>
```

DC.date.issued

Datum der Veröffentlichung der Ressource oder ihres Inhalts.

```
<meta name="DC.date.issued" content="2006-07-15"/>
```

DC.date.modified

Änderungsdatum der Ressource oder von Teilen ihrer Inhalte.

```
<meta name="DC.date.modified" content="2006-09-30T12:30:00"/>
```

DC.description

Eine kurze Inhaltsbeschreibung der Ressource im Umfang von etwa 100 bis 200 Zeichen, wie sie von Suchmaschinen genutzt werden soll.

```
<meta name="DC.description"
content="&Uuml;bersicht und Vergleich aktueller Webeditoren,
Darstellung der Funktionen, Screenshots und Bezugsquellen"/>
```

DC.description.tableOfContents

Auflistung thematischer Unterpunkte der Ressource in Form eines Inhaltsverzeichnisses .

```
<meta name="DC.description.tableOfContents"
 content="Einleitung, &Uuml;bersicht, Details der Editoren,
 Bezugsquellen"/>
```

DC.description.abstract

Eine zusammenfassende Inhaltsangabe der Ressource (jedoch kürzer als »DC.description«.

```
<meta name="DC.description.abstract"
 content="Webeditoren, Programme zur Erstellung von HTML-Dokumenten,
&Uuml;bersicht und Vergleich, Bezugsquellen."/>
```

DC.format

Stichwort(e) für das Genre, dem die Ressource inhaltlich zuzuordnen ist.

```
<meta name="DC.format" content="Programme, Webeditoren"/>
```

DC.format.extent

Beschreibung der Größe der Ressource oder (beispielsweise für Videodaten) ihrer zeitlichen Dauer.

```
<meta name="DC.format.extent" content="120 KB"/>
```

```
<meta name="DC.format.extent" content="8.5 min"/>
```

DC.format.medium

Die Art der Ressource.

```
<meta name="DC.format.medium" content="Webseite"/>
```

Hierbei ist der DC-Schematyp IMT (Internet Media Type) einsetzbar:

```
<meta name="DC.format.medium"
 scheme="IMT" content="text/html"/>
```

DC.identifier

Dient als eindeutige Bezeichnung der Ressource.

```
<meta name="DC.identifier"
 content="http://www.editoren.de/index.html"/>
```

Dies kann durch den DC-Schematypen-URI verdeutlicht werden:

```
<meta name="DC.identifier" scheme="URI"
 content="http://www.editoren.de/index.html"/>
```

DC.language

Bezeichnet die Sprache des Inhalts der Ressource.

```
<meta name="DC.language" content="de"/>
```

Mit dem DC-Schema RFC3066 wird die genaue Form des Kürzels festgelegt:

```
<meta name="DC.language" scheme="RFC3066" content="en"/>
```

DC.publisher

Name und/oder E-Mail-Adresse der für die Veröffentlichung der aktuellen Ressource verantwortlichen Person, Organisation oder Institution

```
<meta name="DC.publisher" content="Curt Coder"/>
```

```
<meta name="DC.publisher"
 content="Curt Coder, curt@editoren.de"/>
```

DC.relation

Die aktuelle Ressource steht in Beziehung zur angegebenen Ressource, die per URL oder durch eine andere Kennzeichnung benannt werden kann.

```
<meta name="DC.relation"
 content="http://www.editoren.de/partner.html"/>
```

DC.relation.isVersionOf

Die aktuelle Ressource stellt eine Abwandlung oder Veränderung der referenzierten Ressource dar.

```
<meta name="DC.relation.isVersionOf"
 content="/webeditoren/uebersicht_kurz.html"/>
```

DC.relation.hasVersion

Die aktuelle Ressource beinhaltet eine Abwandlung oder Veränderung der referenzierten Ressource.

```
<meta name="DC.relation.hasVersion"
 content="/webeditoren/uebersicht_lang.html"/>
```

DC.relation.isReplacedBy

Die aktuelle Ressource wurde durch die referenzierte Ressource ersetzt.

```
<meta name="DC.relation.isReplacedBy"
 content="webeditoren_neu.html"/>
```

DC.relation.replaces

Die aktuelle Ressource dient als Ersatz der referenzierten Ressource.

```
<meta name="DC.relation.replaces"
 content="webeditoren_alt.html"/>
```

DC.relation.isRequiredBy

Die aktuelle Ressource wird von der referenzierten Ressource benötigt.

```
<meta name="DC.relation.isRequiredBy"
 content="webeditoren/index.html"/>
```

DC.relation.requires

Die aktuelle Ressource benötigt die referenzierte Ressource zur Unterstützung oder besseren Darstellung der Inhalte.

```
<meta name="DC.relation.requires"
 content="/webeditoren/screenshots.html"/>
```

DC.relation.isPartOf

Die aktuelle Ressource wird von der referenzierten Ressource einbezogen.

```
<meta name="DC.relation.isPartOf"
 content="/ausbildung.html"/>
```

DC.relation.hasPart

Die aktuelle Ressource bezieht die referenzierte Ressource logisch ein.

```
<meta name="DC.relation.hasPart"
 content="/ausbildungsangebote.html"/>
```

DC.relation.isReferencedBy

Die Ressource wird von der angegebenen Quelle zitiert oder referenziert.

```
<meta name="DC.relation.isReferencedBy"
 content="/webeditoren.html"/>
```

DC.relation.references

Die aktuelle Ressource referenziert oder zitiert die angegebene Quelle.

```
<meta name="DC.relation.references"
 content="/webeditoren/details.html"/>
```

DC.relation.isFormatOf

Die aktuelle Ressource entspricht der referenzierten Originalressource, liegt jedoch in einem, von dieser abweichenden Format vor.

```
<meta name="DC.relation.isFormatOf"
 content="/webeditoren.doc"/>
```

DC.relation.hasFormat

Die aktuelle Originalressource verweist auf eine inhaltsgleiche Kopie eines anderen Speicherformats.

```
<meta name="DC.relation.hasFormat"
 content="/webeditoren.pdf"/>
```

DC.rights

Angaben zu Urheberrechten oder weiteren Bestimmungen, wobei auch auf eine Webadresse verwiesen werden kann.

```
<meta name="DC.rights" content="/copyright.html"/>
```

DC.source

Verweis auf die Quelle der Inhalte (Webadresse, Buch etc.).

```
<meta name="DC.source"
 content="http://www.webeditoren.de/index.html"/>

<meta name="DC.source"
 content="Shakespeares Werke, Berlin 1856"/>
```

DC.subject

Kurze Inhaltsbeschreibung, meist in Form von Schlag- und Stichworten.

```
<meta name="DC.subject" content="Editoren, HTML, Webdesign"/>
```

DC.title

Titel der Webseite; spiegelt inhaltlich meist den `<title>`-Tag wider.

```
<meta name="DC.title" content=" Editoren f&uuml; Webseiten"/>
```

DC.title.alternative

Alternativtitel zur Angabe in `DC.title`, beispielsweise eine Übersetzung oder Abkürzung des Titels (eine zusätzliche Sprachangabe ist möglich).

```
<meta name="DC.title" xml:lang="de"
 content="Editoren f&uuml; Webseiten"/>

<meta name="DC.title.alternative"
 content="Webeditoren"/>

<meta name="DC.title.alternative" xml:lang="en"
 content="Editors für web pages"/>
```

DC.type

Das inhaltliche Genre der aktuellen Ressource.

```
<meta name="DC.type" content="Webeditoren"/>
<meta name="DC.type" content="Ausbildung; Fortbildung"/>
```

Praxistipp

Dublin-Core-Meta-Daten sind vor allem dort nützlich, wo verarbeitende Programme den Meta-Daten vertrauen können, beispielsweise innerhalb von Intranets. Aufgrund aktuellen Missbrauchs werden sie von Suchmaschinen derzeit nicht bevorzugt beachtet – was sich in Zukunft wieder ändern mag.

Die Zukunft des <meta>-Elements

In XHTML 2.0 wird das Element als Container spezifiziert werden, was Textinhalte erlaubt und gewöhnlich das `content`-Attribut ersetzt. Im Rahmen der Spezifikation soll jedoch jedes HTML-Element in die Lage versetzt werden, Meta-Informationen zu tragen. Zu diesem Zweck dient das »Metainformation Attributes Module«, das als Teil der Common Collection von XHTML 2 allen Elementen zugewiesen wird. Es umfasst sechs Attribute, darunter in bekannten Rollen `rel`, `rev`, `content` und `property`. Im Vorschlag der WHAT WG für »Web Applications 1.0« bleibt `<meta>` hingegen in seiner gegenwärtigen Form als leeres Element erhalten.

14.56 nextid

Historisch: Das Element `<NEXTID>` sollte Editorsoftware als Hilfestellung dienen, um innerhalb des Dokuments automatisiert Ankerpunkte zu erstellen, die als Ziel von Hyperlinks dienen können.

14.56.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element, Head
XHTML-Modul	– (abgeschafft)
Kontext	im `<head>`-Container
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML Level 0; HTML 1.0, 2.0 (deprecated); **abgeschafft** ab HTML 3.2
Mobil-Spezifikationen	– (nirgends spezifiziert)

Syntaxbeispiel:

```
<NEXTID N=z123>
```

Anmerkung: Das Setzen von Anführungszeichen um den Attributwert war in der Spezifikation für das Element nicht vorgesehen.

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

–

14.56.2 Attribute (Übersicht)

Standardattribute

HTML	–

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

W3C	`n`

14.56.3 Offizielle Attribute – W3C

n

Wert	*cdata*
Typ	Eine alphanumerische Zeichenkette
Verwendung	**Obligatorisch**

Das obligatorische `n`-Attribut stellt einen Wert zur Verfügung, der als Startwert (nächstes freies Label) für automatisch generierte, SGML-kompatible Identifier für neue Anker-Elemente innerhalb des Dokuments dienen kann.

14.56.4 Erläuterung zum Element

Das <NEXTID>-Element war als Unterstützung bei der Automatisierung der Erstellung von Webseiten durch hierfür eingerichtete Editoren (*automatic hypertext editors*) konzipiert. Es richtete sich daher weder an User Agents (lt. Spezifikation HTML 2.0: »Browser software may ignore this tag!«) noch sollte es von Webprogrammierern im Rahmen handkodierter Seiten eingesetzt werden.

Hintergedanke war, dass ein HTML-Editor in der Lage sein sollte, innerhalb des Dokuments automatisch SGML-ID-kompatible[42] Ankerpunkte (Name-Anker) zu setzen , die dokumentweit einzigartig sein sollten. Um dies zu ermöglichen, konnte dem Editor mittels des n-Attributs des <NEXTID>-Tags ein Startwert der Form »z123« übergeben werden, von dem ausgehend neue Ankerpunktnamen generiert werden durften. Es sollte nicht nur eine potentielle Dopplung von Identifiern verhindert werden, sondern laut Berners-Lee auch das versehentliche Wiederverwenden von gelöschten IDs.

Kurzzeitig im Gespräch, jedoch nie umgesetzt, war eine Erweiterung von <NEXTID>, um eine Versionierung bzw. Checkin/Checkout von Dokumenten zu unterstützen. Stattdessen wurde das Element ohne nachhaltige Unterstützung durch entsprechende Editoren noch eine Weile in den offiziellen Spezifikationen mitgeschleppt, wobei es bereits in HTML 2.0 als »deprecated« galt und nicht mehr verwendet werden sollte. Mit der Spezifikation HTML 3.2 verschwand <NEXTID> schließlich endgültig.

14.57 noframes

Das <noframes>-Element stellt innerhalb eines Framesetdokuments alternative Inhalte für nicht-framefähige User Agents bereit.

14.57.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Strukturcontainer
XHTML-Modul	Frames
Kontext	Als letztes Element im äußeren <frameset>-Container eines Framesetdokuments

42 Der Bezeichner darf demnach aus alphanumerischen Zeichen bestehen, jedoch nicht mit einer Ziffer beginnen. Man entschied sich für eine Ziffernfolge mit einem vorangestellten Buchstaben.

Erlaubte Inhalte	<body>-Element; darin Block und Inlineelemente in Form eines alternativ zu zeigenden Dokuments
Offizielle W3C Spezifikationen	HTML 4.0/4.01 (tr/fs); XHTML 1.0 (tr/fs); **nicht enthalten** in HTML 4.0/4.01 strict XHTML 1.0 strict, XHTML 1.1, XHTML 2.0
Mobil-Spezifikationen	– (nirgends spezifiziert)

Syntaxbeispiel:

```
<noframes> ... </noframes>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

–

14.57.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onreadystatechange

Elementspezifische Attribute

W3C	–

14.57.3 Erläuterung zum Element

Das <noframes>-Element dient als Container für Alternativinhalt, der von User Agents ausgewertet werden soll, die keine Frames interpretieren. Es ist Teil der »transitional«- und der »frameset«-DTDs von HTML 4.01 und XHTML 1.0. In den »strict«-DTDs ist das Element nicht enthalten (daher auch nicht in XHTML 1.1).

Inhaltsmodell

Das Inhaltsmodell des <noframes>-Containers variiert dabei je nach Kontext (die Definitionen für HTML sind analog):

```
<!-- XHTML "frameset" DTD: -->
<!ELEMENT noframes (body)>
```

In der »frameset«-DTD muss <noframes> das <body>-Element enthalten.

Die Definition für die »transitional«-DTD ist wesentlich liberaler:

```
<!-- XHTML "transitional" DTD: -->
<!ELEMENT noframes %Flow;>
```

Hierbei umfasst das Parameter-Entity %Flow praktisch alle Block- und Inlineelemente (nicht jedoch <body>, da <noframes> in diesem Falle innerhalb des <body>-Elements stehen muss), sogar <noframes> selbst[43] – wobei eine Verschachtelung von <noframes> jedoch wohl wenig Sinn ergibt. Der erlaubte Inhalt von <noframes> ist in diesem Kontext also praktisch beliebig.

Auftreten im Frameset-Kontext

Tritt das <noframes>-Element in Frameset-Dokumenten auf, die demzufolge der »frameset«-DTD entsprechen, so darf *maximal ein* <noframes>-Container auftreten, der als obligatorisches Kindelement ein <body>-Element als Alternativinhalt zum Frameset enthält. In der Regel wird <noframes> in diesem Kontext eingesetzt, um Suchmaschinen oder Screenreadern direkten Zugriff auf die frameeigenen Inhaltsseiten oder die im Frameset verwendete Navigation zu verschaffen.

Im Gegensatz zu verbreiteten Gepflogenheiten ist es in diesem Falle nicht korrekt, den <noframes>-Container *hinter* den äußeren <frameset>-Container zu stellen – er muss hier vielmehr innerhalb von diesem als *letztes Kindelement* auftreten und muss ein <body>-Element enthalten:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>XHTML-Frameset mit Noframes-Bereich</title>
</head>
<frameset cols="120,*">
   <frame src="navi.html" name="nav"/>
   <frame src="inhalt.html" name="inhalt"/>
   <noframes>
```

43 Dies erlaubt dem <noframes>-Container wiederum, überall als erlaubter Inhalt aufzutreten.

```
      <body>
      <!-- Link zur Navigationsseite für Suchmaschinen: -->
      <p><a href="navi.html">Sitenavigation</a></p>
      <!-- Links zu Inhaltsdokumenten für Suchmaschinen: -->
      <p><a href="inhalt.html">Inhaltsbeschreibung</a></p>
      <p><a href="inhalt_2.html">Inhaltsbeschreibung</a></p>
      </body>
   </noframes>
</frameset>
</html>
```

Auftreten im Dokumentkontext

Das Auftreten eines `<noframes>`-Containers in normalen HTML-Dokumenten ist ebenfalls gestattet, sofern diese einer »transitional«-DTD entsprechen. Die Häufigkeit und das für `<noframes>` erlaubte Inhaltsmodell ist dabei nur wenig eingeschränkt: Das Element darf praktisch in unbegrenzter Anzahl in beliebigen Elementcontainern auftreten und dabei beliebige Inhalte tragen.

Auch hier wird der Container nur von User Agents ausgewertet, die keine Frames darstellen – beispielsweise kann so auf Inhaltsseiten eines Framesets eine Hilfsnavigation für den Fall eingefügt werden, dass diese Seite außerhalb ihres Framekontexts und durch einen »unvorhergesehenen« User Agent betrachtet wird:

```
<!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>Frameinhaltsseite mit Hilfsnavigation</title>
</head>
<body>
<!-- regulärer Dokumentinhalt -->
<h1>Hier ist noframes-Inhalt auf der Seite</h1>
<p>Aber es gibt auch normale Elementcontainer mit Text.</p>
...
<noframes>
   <!-- dies ist für framefähige User Agents unsichtbar: -->
   <p>
   <a href="inhalt1.html">Inhalt 1</a> |
   <a href="inhalt2.html">Inhalt 2</a> |
   <a href="inhalt3.html">Inhalt 3</a> |
   <a href="inhalt4.html">Inhalt 4</a>
   </p>
</noframes>
```

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

Wie dieses Dokument in einem nicht framefähigen Textbrowser wie Lynx dargestellt wird, ist in Abbildung 14.62 zu sehen. In anderen Browsern wird die Navigation nicht gezeigt.

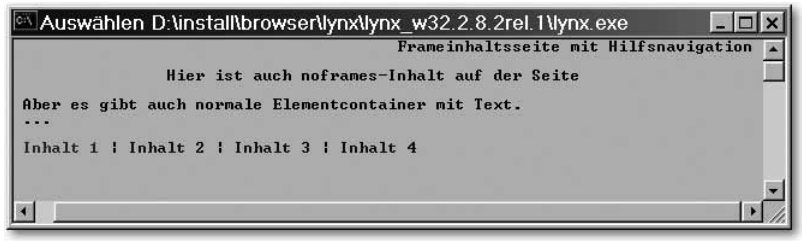

Abbildung 14.62 Hilfsnavigation in <noframes>-Container (Lynx)

Barrierefreiheit

Der Einsatz von `<noframes>`-Bereichen erleichtert die Lesbarkeit der Website für Screenreader und andere User Agents, die keine Frames unterstützen. Hierdurch ist gleichzeitig die Anforderung erfüllt, dynamische Inhalte zugänglich zu halten (vergleiche WAI 6.5, BITV 6.5).

Weitere Informationen über den `<frameset>`-Container finden Sie in Abschnitt 14.35, *frameseb*., solche über das `<frame>`-Element in Abschnitt 14.35, *frame*.

14.58 noscript

Der `<noscript>`-Container enthält Inhalte, die dann dargestellt werden, wenn der User Agent kein clientseitiges Scripting unterstützt oder dieses deaktiviert ist.

14.58.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Scripting
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1; **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	– (nirgends spezifiziert)

Syntaxbeispiel:

```
<noscript> ... </noscript>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

14.58.2 Attribute (Übersicht)

Standardattribute

HTML	Class, dir, id, lang, style, title
XHTML	Class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onreadystatechange

Elementspezifische Attribute

W3C	–

14.58.3 Erläuterung zum Element

Das Element stellt alternative Inhalte zur Verfügung, wenn dem Client die Ausführung von Scripten generell nicht zu Verfügung steht. Dies ist dann der Fall, wenn entweder überhaupt keine Scriptsprache unterstützt wird oder der Scriptinterpreter (beispielsweise aus Sicherheitsgründen) deaktiviert ist.

Es ist dabei *nicht möglich*, einen <noscript>-Container *unmittelbar* mit einem bestimmten <script>-Container zu verknüpfen und an dessen Ausführbarkeit zu binden: <noscript>-Container werden stets global und dokumentweit ausgeführt.

Beispielsweise kann ein <noscript> nicht als Alternative eines in VisualBasic (oder einer anderen, nicht unterstützten Sprache) geschriebenen Scriptcontainers aktiviert werden, wenn die Seite in einem User Agent betrachtet wird, der JavaScript ausführen kann und dieses aktiviert ist.

Als Teil des <head>-Contents ist der <noscript>-Container nicht gestattet, kann also dort nicht als Alternative zu Scriptblöcken eingesetzt werden. Das folgende Beispiel ist daher nicht legal (und funktioniert auch nicht zuverlässig):

```
<!-- illegale Weiterleitung auf scriptfreie Seite: -->
<noscript>
   <meta http-equiv="refresh"
    content="1; url=/kein_script.html">
</noscript
```

Ganz nebenbei wird auch gegen das Inhaltsmodell von <noscript> verstoßen, das nur Blockelemente zulässt.

[zB] *Beispiel:*

```
<p>Hier kommt das erste Script (JavaScript), dem ein
&lt;noscript&gt;-Block folgt:</p>

   <script type="text/javascript">
   <!--
   document.write('<p>Script 1: Bin schon fertig!</p>');
   //-->
   </script>

<noscript><p>Um dies hier zu sehen, m&uuml;ssen Sie JavaScript
deaktivieren. </p></noscript>

<p>Zwischen Scriptbl&ouml;cken kann statisches HTML stehen. Jetzt
kommt ein <b>nicht ausf&uuml;hrbarer</b> Scriptblock in der Sprache
&quot;JojoScript&quot;, dem ebenfalls ein &lt;noscript&gt;-Block
folgt:</p>

   <script type="text/jojoscript">
   <!--
   echo_out('Bingo! Ich kann JojoScript!');
   -->
   </script>

<noscript><p>Diese Sprache kenne ich nicht. Leider
&quot;geh&ouml;rt&quot; dieser Block nicht zu dem vorigen
Scriptblock, sonst w&uuml;rde er vielleicht gezeigt.</p></noscript>
```

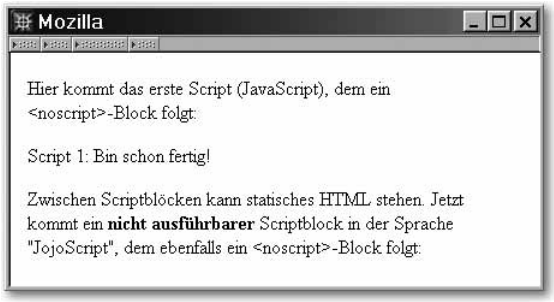

Abbildung 14.63 Ist das Scripting im Browser aktiviert, so wird kein <noscript> ausgeführt.

Der Browser (siehe Abbildung 14.63) führt den ersten Scriptcontainer aus, nicht jedoch den zweiten, da die für ihn verlangte Scriptsprache nicht unterstützt wird. Es wird kein `<noscript>`-Container ausgewertet, da Scripting grundsätzlich zugelassen und aktiviert ist. Ist das Scripting deaktiviert (siehe Abbildung 14.64), werden hingegen die Inhalte *aller* `<noscript>`-Container dargestellt.

Abbildung 14.64 Bei deaktiviertem Scripting zeigt der Browser alle <noscript>-Container.

Anmerkung: In XHTML 2.0 ist dieses Problem behoben, da hier anstelle der `<script>`- und `<noscript>`-Container ein verschachtelbarer `<handler>`-Container vorgesehen ist. Dieser wird (vergleichbar mit dem `<object>`-Container) von außen nach innen abgearbeitet. wobei zum weiter innen liegenden `<handler>` übergegangen wird, sobald der äußere nicht ausgeführt werden kann.

In XHTML 2.0 wird also Folgendes möglich sein:

```
<handler type="text/x-vbscript">
   ' hier steht VB-Script
   <handler type="text/javascript">
      // hier steht JavaScript, das ausgeführt wird,
```

```
      // sobald die Sprache des äußeren Handlers
      // nicht unterstützt wird
   </handler>
</handler>
```

14.59 object

Das <object>-Element dient zum Einbetten eines beliebigen Datenobjekts (z. B. Video, Audio, Flash, Java-Applet) in ein HTML-Dokument.

14.59.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Object
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	<object>-, <param>-Elemente sowie andere Block- und Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<object> ... </object>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
object     { display: inline; }
```

14.59.2 Attribute (Übersicht)

Standardattribute

HTML	Class, dir, id, lang, style, title
XHTML	Class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforeeditfocus`, `onbeforeupdate`, `onblur`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onerror`, `onerrorupdate`, `onfocus`, `onload`, `onlosecapture`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onrowsdelete`, `onrowsinserted`, `onscroll`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `archive`, `border`, `classid`, `codebase`, `codetype`, `data`, `declare`, `height`, `hspace`, `name`, `standby`, `type`, `usemap`, `vspace`, `width`
Internet Explorer	`code`, `notab`[44], `object` *Access-, Databinding-, Edit-Attribute*

14.59.3 Offizielle Attribute – W3C

align

Wert	*"top"* | *"middle"* | *"bottom"* | *"left"* (float) | *"right"* (float)
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut erlaubt es, ein eingebettetes Objekt so wie ein Image im Zeilenkontext (`"top"`, `"middle"`, `"bottom"`) oder mittels der Floatattributwerte `"left"` und `"right"` am entsprechenden Rand des übergeordneten Containers zu positionieren. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

Zusätzlich zu den Werten der Spezifikation unterstützen einige Browser auch die Werte `"center"`, `"baseline"`, `"absbottom"`, `"absmiddle"` und `"texttop"`.

44 Abgeschafft in IE4.0.

archive

Wert	*URL [, URL2, URL3, ... URLn]*
Typ	ein oder mehrere URI-Strings
Verwendung	Optional

Das `archive`-Attribut kann den absoluten oder relativen URL einer alternativen, gepackten Version des Objekt (GNU-Zip-File, unkomprimiert) nennen, um den Ladevorgang zu beschleunigen. Zusätzlich können die URLs weiterer zu ladender Klassen angegeben werden.

border

Wert	*number*
Typ	eine Ganzzahl größer oder gleich 0
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `border`-Attribut gibt die Dicke des um das Objekt zu zeichnenden Rahmens in Pixel an. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

classid

Wert	*URL*
Typ	ein URI-String oder eindeutiger Bezeichner
Verwendung	Optional

Das `classid`-Attribut bezeichnet das Objekt über einen URI oder einen anderen eindeutigen Bezeichner. In Falle eines Java-Applets dient das Attribut dazu, die zur Ausführung erforderliche JVM (Java Virtual Machine) zu bezeichnen.

```
classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
```

codebase

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional

Das `codebase`-Attribut gibt einen absoluten oder relativen URL an, der als Basis-URL für den URL des `code`-Attributs dient. Ein relativer URL wird als bezogen auf den URL des Dokuments oder, falls ein solcher vorhanden ist, auf den `href`-Wert des `<base>`-Tags ausgewertet.

```
codebase="http://java.sun.com/products/plugin/autodl/..."
```

codetype

Wert	*cdata*
Typ	ein Media-Typ-Bezeichner
Verwendung	Optional

Das `codetype`-Attribut teilt dem Browser den Internet Media Type des zu ladenden Objekts mit. Ist der Media Type dem Browser unbekannt, kann er das Laden der Ressource abbrechen. Das Attribut ist nur relevant, wenn gleichzeitig das `classid`-Attribut gesetzt ist.

data

Wert	*URL* | *#hexcode*
Typ	ein URI-String (oder hexadez. Programmquellcode)
Verwendung	Optional

Das `data`-Attribut übergibt einen URI-String, der die Datenquelle bezeichnet, die die Quelldaten des Objekts zur Verfügung stellt. Alternativ ist auch die Möglichkeit gegeben, hier unmittelbar Quellcode in Form einer Zeichenkette zu übergeben, die als Hexadezimalwerte interpretiert wird.

declare

Wert	*"declare"*
Typ	fesgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Mittels des `declare`-Attributs wird die Ressource des Objekts zwar geladen, jedoch nicht unmittelbar initialisiert. Die Initialisierung geschieht erst durch nachfolgenden Aufruf mittels des `id`. Dies kann von Bedeutung sein, wenn das Objekt später im Dokument referenziert wird, um ihm beispielsweise Daten zur Verfügung zu stellen (bedeutsam in iMode-Anwendungen).

```
<object declare="declare" classid="java:minesweeper.class"
 id="minesweeper" codetype="application/java"></object>

<p><a href="#minesweeper">Minesweeper starten!</a></p>
```

Hinweis: Das Attribut wird in Opera nicht unterstützt. **[«]**

height

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `height`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für die Höhe des für das Objekt freizuhaltenden Raumes.

Anmerkung: Beachten Sie, dass dies ebenso wie beim `width`-Attribut in erster Linie die Abmessungen des Viewports festlegt, innerhalb dessen das Objekt gezeigt wird. Internet Explorer zeigt im Objektbereich Scrollbalken, wenn die angegebene Fläche zu klein ist (siehe Abbildung 14.65). Mozilla und Opera hingegen skalieren das eingebettete Objekt.

hspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `hspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den symmetrisch links und rechts vom Objekt freizuhaltenden Raum. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

name

Wert	*nmtoken*
Typ	Zeichenkette in Form eines Nametokens
Verwendung	Optional

Das `name`-Attribut nennt eine Zeichenkette, die von Scripten als Bezeichner für das Objekt verwendet werden kann.

[»] *Hinweis:* Bevorzugt sollte für diesen Zweck das `id`-Attribut eingesetzt werden. Wird (entgegen der Spezifikation) innerhalb des `<object>`-Containers zusätzlich ein `<embed>` eingesetzt, so dürfen beide Elemente keinesfalls identische `name`- oder gar `id`-Attributwerte erhalten. Setzen Sie in diesem Fall für `<object>` ein `id`- und für `<embed>` ein gleichnamiges `name`-Attribut.

standby

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Dem `standby`-Attribut kann eine beliebige Zeichenkette übergeben werden, die angezeigt wird, während das Objekt geladen wird. Das Attribut ist *case-sensitive*.

Hinweis: Das Attribut wird in Opera nicht unterstützt. **[«]**

tabindex

Wert	*number*
Typ	Eine Ganzzahl
Verwendung	Optional

Barrierefreiheit. Die mittels des `tabindex`-Attributs übergebene Ganzzahl legt, in aufsteigender Richtung, die Position des Objekt-Elements in der Tabulatorwahl-Reihenfolge (⇆-Taste) fest.

Die Tab-Reihenfolge gestaltet sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes `tabindex`-Attribut in Quelltextreihenfolge (hauptsächlich Links).

Anmerkung: Ein negativer Tabindex-Wert entfernt das Objekt im IE vollständig aus der Tab-Reihenfolge. Die anderen Browser ordnen es in die normale Tab-Reihenfolge ein, als ob kein `tabindex`-Attribut gesetzt wäre.

type

Wert	*cdata*
Typ	Ein MIME-Typ-Bezeichner
Verwendung	Optional

Das `type`-Attribut nennt den MIME-Typ der als Objekt zu ladenden Ressource.

usemap

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `usemap`-Attribut adressiert über einen (in der Regel lokalen) URI eine mit dem Objekt zu verwendende Imagemap-Definition `<map>`. Für eine lokale Definition besitzt der URI die Form eines Fragment-Identifiers, der den Wert des `name`-Attributs des `<map>`-Elements verwendet.

```
<object usemap="#map_bezeichner"> ... </object>
```

vspace

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut). Das `vspace`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für den über und unter dem Objekt freizuhaltenden Raum. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

width

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `width`-Attribut nennt einen Pixelwert (Ganzzahl) oder eine Prozentangabe für die Breite des für das Objekt zu reservierenden Raums.

14.59.4 Proprietäre Attribute – Internet Explorer

code

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `code` nennt den absoluten oder relativen URL der zu ladenden Ressource (z. B. einer Javaklasse). Alternativ kann die Einbindung auch über das `archive`- oder im Internet Explorer über das `object`-Attribut erfolgen.

notab

Wert	–
Typ	Leeres Marker-Attribut
Verwendung	Optional
Gültigkeit	IE3 (abgeschafft seit IE4)

(Historisch) Das `notab`-Attribut diente dazu, das Objekt aus der Tab-Reihenfolge des Internet Explorers zu entfernen. Für das gewünschte Ergebnis kann dort heutzutage das `tabindex`-Attribut auf den Wert –1 gesetzt werden.

object

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das `object`-Attribut nennt einen URL, unter dem die als Objekt zu ladende Ressource in serialisierter Form zu finden ist (im Falle eines Java-Applets die Klasseninformation und der gespeicherte Zustand des Applets als Bytefolge).

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.59.5 Erläuterung zum Element

Das Element `<object>` dient zum Einfügen eines beliebigen Medienobjektes in eine HTML-Seite. Es stellt damit den »offiziellen« Gegenentwurf des W3C zum proprietären Netscape-Element `<embed>` dar, für das keine Spezifikation existiert. Im Gegensatz zu `<embed>` ist `<object>` kein leeres Element, darf also Inhalte besitzen, die alternativ zum einzubettenden Medienobjekts dargestellt werden können. Als Inlinecontainer darf `<object>` nicht direkt in `<body>` stehen, sondern muss stets innerhalb eines Blockcontainers eingebettet werden.

Auch an Stelle des `<img>`-Tags kann `<object>` eingesetzt werden, was auch hier alternative Inhalte ermöglicht. In XHTML 2.0 wird vorgeschlagen, zum Einbinden von Medien (was ausdrücklich auch statische Bilder beinhaltet) grundsätzlich `<object>` einzusetzen – das `<img>`-Element ist dort nur noch aus Gründen der Abwärtskompatibilität vorhanden, allerdings ebenfalls zum Container erweitert.

Darstellbarkeit eines eingebetteten Objekts

Über die Darstellbarkeit eines Objekts entscheidet der Browser anhand des MIME-Typs, der ihm durch das `type`-Attribut mitgeteilt wird. Im folgenden Beispiel ist der MIME-Typ `image/jpeg` bekannt. Der Browser wird daher das Objekt anzeigen, nicht das im Inneren des `<object>`-Containers befindliche Menü:

```
<object height="155" width="155" data="eiffelturm.jpg"
 type="image/jpeg">
    <p><b>Fallbackmen&uuml;:</b></p>
    <ul>
    <li><a href="paris_1.html">Klicken Sie hier</a></li>
    <li><a href="paris_2.html">... oder da</a></li>
    </ul>
</object>
```

Abbildung 14.65 Bilder mit <object> einzubinden, kann schief gehen (IE)

Sobald der MIME-Typ unbekannt oder aus anderen Gründen nicht ausführbar ist, greift der Browser auf den Inhalt des Containers zu. Im folgenden Fall wird durch `type` ein unbekannter (nicht existenter) Applikationstyp `application/unbekannt` gefordert:

```
<object height="310" width="310" data="unbekannt.ubk"
 type="application/unbekannt">
    <p><b>Fallbackmen&uuml;:</b></p>
    <ul>
    <li><a href="paris.html">Klicken Sie hier</a></li>
    <li><a href="paris.html">... oder da</a></li>
    </ul>
</object>
```

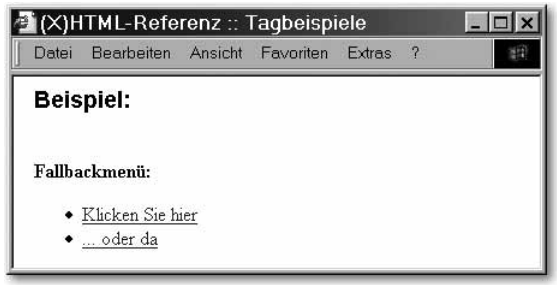

Abbildung 14.66 Ist die geforderte Applikation nicht verfügbar, so wird der Elementinhalt angezeigt.

Verschachtelung und Alternativcontent

Eine Verschachtelung mehrerer `<object>`-Container ist möglich. Dies erlaubt es, in mehreren Stufen auf alternative Inhalte auszuweichen, sobald ein Inhaltstyp nicht unterstützt wird.

```
<!-- 1.: Versuch, ein Java-Applet einzubinden -->
<object type="application/x-java-applet"
src="http://www.beispiel.de/applets/beispiel.class">

    <!-- 2.: Alternativ wird ein Video gezeigt -->
    <object type="video/mpeg"
     src="http://www.beispiel.de/videos/beispiel.mpeg">

        <!-- 3.: Sonst wird ein Standbild verwendet -->
        <object type="image/png"
        src="http://www.beispiel.de/images/beispiel.png">

            <!-- 4.: Zuallerletzt ein Beschreibungstext -->
```

```
            Dies ist die Darstellung eines Beispiels.

        </object>
    </object>
</object>
```

Browserkompatibler Einsatz von <object>

Auch im Dokumentkopf dürfen `<object>`-Container auftreten – allerdings muss feststehen, dass sie in diesem Fall keinen visuellen Output erzeugen oder darzustellenden Inhalt besitzen. Auf diesem Wege kann beispielsweise das Einbinden eines Soundfiles für Hintergrundmusik erfolgen. Folgendes Beispiel funktioniert in Mozilla-Browsern, jedoch nicht im Internet Explorer:

```
...
<head>
   ...
   <object type="audio/x-mpeg" data="meinsound.mp3">
   <!-- hier darf kein darstellbarer Inhalt stehen -->
   </object>
</head>
...
```

Der Internet Explorer steuert die Objektdarstellung gewöhnlich mittels Active-X (was vielerorts als Sicherheitsrisiko angesehen wird). Hierfür wird über das `classid`-Attribut der CLSID-Schlüssel angegeben, wie er in der Windows-Registry abgelegt ist. Laut Microsoft hat ein dort registrierter **ActiveX-Class-ID** die folgende allgemeine Form:

```
clsid:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Dessen Wert ruft die zur Ausführung des Objekts erforderliche Active-X-Klasse auf. Im folgenden Beispiel handelt es sich wieder um ein Soundfile. Beachten Sie, dass das `type`-Attribut vom Internet Explorer nicht benötigt wird, wenn die Applikationszuordnung über einen CLSID erfolgt:

```
...
<head>
   ...
   <object
    classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
      <param name="FileName" value="meinsound.mp3" />
   </object>
</head>
...
```

Der Name der abzuspielenden Sounddatei wird über einen im `<object>`-Container befindlichen `<param>`-Tag angegeben. Diese Lösung arbeitet allerdings nicht in Mozilla-Browsern.

Die einfache Hinzunahme des `type`- und `data`-Attributs genügt jedoch nicht, da sich Mozilla am Vorhandensein des `classid`-Attributs stört. Entfernt man dieses, so wird der Sound in Mozilla abgespielt und ebenfalls noch im Internet Explorer, der in Abwesenheit von `classid` auf das `type`-Attribut zugreift, aber noch immer den Dateiverweis aus `<param>` benötigt.

Eine browserübergreifende Lösung verwendet `type`- und `data`-Attribut und gibt die Datei zusätzlich mittels `<param>`-Element an:

```
<head>
   ...
   <object  type="audio/x-mpeg"  data="meinsound.mp3">
       <param name="FileName" value="meinsound.mp3" />
       <!-- hier darf kein darstellbarer Inhalt stehen -->
   </object>
</head>
...
```

14.60 ol

Logisches Element `"ordered list"`. Der `<ol>`-Container erzeugt eine so genannte geordnete Liste, die Listenelemente `<li>` durch Nummerierung oder vergleichbare Aufzählungselemente (Buchstaben, röm. Ziffern) kennzeichnet.

14.60.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	List
Kontext	Erlaubt innerhalb von `<applet>`, `<blockquote>`, `<body>`, `<button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<map>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`
Erlaubte Inhalte	Listenelemente `<li>`
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<ol> ... </ol>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
ol   { display: block;
       list-style-type: decimal;
       margin-left: 40px;
  /*   alternatives Margin:   */
  /*   margin-left: 4em;      */
       }
```

14.60.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	compact, start, type
Internet Explorer	Access-, Edit-Attribute

14.60.3 Offizielle Attribute – W3C

compact

Wert	*"compact"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `compact`-Attribut reduziert den Abstand zwischen den Listenelementen der Liste und bewirkt eine kompaktere Darstellung der Liste.

Anmerkung: Das `compact`-Attribut wird in der Praxis von kaum einem Browser unterstützt. Es ist anzuraten, für die beabsichtigte Wirkung auf das CSS-`margin`-Property der Listenelemente zurückzugreifen.

start

Wert	*number*
Typ	Eine Ganzzahl
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `start`-Attribut bestimmt mittels einer Ganzzahl den Anfangswert der Nummerierung der Listenelemente `<li>` innerhalb der geordneten Liste. Die Art des ausgegebenen Nummerierungszeichens ist abhängig vom `type`-Attribut der umgebenden Liste oder des Listenelements selbst.

Hier beginnt die Zählung der Liste mit der Ziffer 3:

```
<ol start="3">
<li>Ein Listenelement</li>
<li>Ein Listenelement</li>
</ol>
```

Hier ist es der Kleinbuchstabe c:

```
<ol type="a" start="3">
<li>Ein Listenelement</li>
<li>Ein Listenelement</li>
</ol>
```

Mittels des `type`-Attributs kann die Ausgabe für einzelne Listenelemente geändert werden. Die eigentliche Nummerierung ist davon nicht betroffen.

Hier beginnt die Zählung der Liste mit der Ziffer 3. Das zweite Listenelement wird wegen seines `type`-Attributs römisch mit »iv« beziffert, das dritte wieder arabisch mit 5:

```
<ol start="3">
<li>Ein Listenelement</li>
<li type="i">Ein Listenelement</li>
<li>Ein Listenelement</li>
</ol>
```

[!] *Achtung:* Ein per `value`-Attribut für ein Listenelement `<li>` (siehe dort) gesetzter Wert hat Vorrang vor der normalen Nummerierung.

type

Wert	*"1"* | *"a"* | *"A"* | *"i"* | *"I"*
Typ	Festgelegte Zeichen, die den Aufzählungstyp bestimmen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `type`-Attribut bestimmt die Art des Nummerierungszeichens des Listenelements `<li>` innerhalb der geordneten Liste. Defaultwert ist die Nummerierung mit arabischen Ziffern. Beim Attributwert wird im Gegensatz zu den meisten anderen Attributen die Groß- und Kleinschreibung beachtet (*case-sensitive*).

- `type="1"` (Default)
  erzeugt eine Nummerierung mit arabischen Ziffern.
- `type="a"`
  erzeugt eine Nummerierung mit Kleinbuchstaben.
- `type="A"`
  erzeugt eine Nummerierung mit Großbuchstaben.
- `type="i"`
  erzeugt eine Nummerierung mit kleinen römischen Ziffern.
- `type="I"`
  erzeugt eine Nummerierung mit großen römischen Ziffern.

Anmerkung: Der Wert des Attributs der eingeschlossenen Listenelemente `<li>` hat stets Vorrang vor der defaultmäßig oder per `type`-Attribut vorgegebenen Nummerierungsform des Listencontainers. Dies ermöglicht auch die punktuelle Zuordnung eines neutralen Bullets (*disc*, *circle* oder *square*) innerhalb einer geordneten Liste `<ol>`. Im Internet Explorer setzt der Versuch, ein artfremdes

Bullet festzulegen das entsprechende Listenelement auf den *Defaulttyp* der umgebenden Liste (*nicht* auf den dort per `type`-Attribut vorgegebenen Typ).

CSS-Alternative: Verwenden Sie das Property `list-style-type`. Neben den oben angegebenen Varianten können Sie auch den Typ `"none"` zuweisen, was das Nummerierungselement komplett unterdrückt.

14.60.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.60.5 Erläuterung zum Element

Durch den `<ol>`-Container werden so genannte »geordnete« Listen erzeugt, bei denen den Listenelementen eine Nummerierung vorangestellt wird. Im einfachsten Fall ist dies eine Nummerierung mit arabischen Zahlen:

```
<ol>
   <li>Listenelement</li>
   <li>Listenelement</li>
   <li>Listenelement</li>
</ol>
```

erzeugt eine Ausgabe der Form

1. Listenelement
2. Listenelement
3. Listenelement

Auf die *Art* der Nummerierung kann mittels des `type`-Attributs von `<ol>` Einfluss genommen werden. Dieses gilt wie die restlichen Attribute von `<ol>` jedoch als »deprecated« und sollte nicht mehr eingesetzt werden.

Das CSS-Property list-style-type

Der korrekte, zeitgenössische Ansatz verwendet CSS und legt die Nummerierung mittels des Properties `list-style-type` fest, was obendrein erweiterte Möglichkeiten bietet. Die wichtigsten, für den `<ol>`-Container in Frage kommenden Werte sind

- `list-style-type: decimal`
  Dezimalzahlen (1., 2., 3., ...)
- `list-style-type: decimal-leading-zero`
  Dezimalzahlen mit führender Null (01.,02., 03., ...)
- `list-style-type: lower-roman`
  Kleine römische Ziffern (i., ii., iii., ...)
- `list-style-type: upper-roman`
  Grosse römische Ziffern (I., II., III., ...)
- `list-style-type: lower-greek`
  Kleine griechische Buchstaben (, , , ...)
- `list-style-type: lower-latin`
  Kleine lateinische Buchstaben (a., b., c., ...)
- `list-style-type: upper-latin`
  Große lateinische Buchstaben (A., B., C., ...)

Des Weiteren ist eine Nummerierung mit hebräischen, georgischen oder armenischen Ziffern möglich. Deren Darstellbarkeit hängt jedoch von den zur Verfügung stehenden Zeichensätzen ab.

[zB] *Beispiel:*

Verschiedene »geordnete« Listen. Beachten Sie, dass das `type`-Attribut in den »strict«-Varianten von HTML und XHTML nicht mehr eingesetzt werden darf.

```
<ol>
   <li>Tomaten</li>
   <li>Basilikum</li>
</ol>

<ol type="a">
   <li>Rosen</li>
   <li>Tulpen</li>
   <li>Nelken</li>
</ol>

<ol type="i">
   <li>Hammer</li>
```

```
   <li>Amboss
      <ol type="a">
      <li>Damaststahl</li>
      <li>Sandwichblech</li>
      <li>Meteor-Eisen</li>
      </ol>
   </li>
</ol>
```

Abbildung 14.67 -Listen mit verschiedenen type-Attributen

14.61 optgroup

Das `<optgroup>`-Element gruppiert `<option>`-Elemente innerhalb eines `<select>`-Containers zu Menügruppen.

14.61.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Forms
Kontext	Innerhalb eines `<select>`-Containers
Erlaubte Inhalte	Elemente `<option>` in beliebiger Anzahl
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile *nicht enthalten* in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<optgroup> ... </optgroup>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

–

14.61.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onlosecapture`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`disabled`, `label`

14.61.3 Offizielle Attribute – W3C

disabled

Wert	*"disabled"*
Typ	Festgelegter Bezeichner
Verwendung	Optional

Das `disabled`-Attribut legt fest, das eine durch ein `<optgroup>`-Element gruppierte Optionsgruppe einer Selectbox deaktiviert ist.

label

Wert	*cdata*
Typ	Eine beliebige Zeichenkette
Verwendung	Optional

Das `label`-Attribut bestimmt den Namen der Optionsgruppe, der bei der Gruppierung von Optionen als Gruppenbezeichner verwendet wird. Das Attribut ist *case-sensitive*.

14.61.4 Erläuterung zum Element

Der `<optgroup>`-Container gruppiert `<option>`-Elemente innerhalb eines `<select>`-Containers zu so genannten Optionsgruppen. Sein `label`-Attribut erzeugt innerhalb der Auswahlliste eine Überschrift für die gebildete Gruppe. Der Container darf nicht mit sich selbst verschachtelt werden, so dass folglich keine Gruppen innerhalb von Optionsgruppen gebildet werden können (das Inhaltsmodell lässt lediglich `<option>`-Elemente zu). Es existiert in der Spezifikation zu HTML 4.01 jedoch ein Hinweis, dass »spätere Versionen von HTML« eine Verschachtelung des Elements erlauben könnten.

Die ursprüngliche Intention des Elements bestand in der Erzeugung verschachtelter (beispielsweise horizontal ausklappbarer) Menülisten ähnlich dem Windows-Startmenü. Diese als »cascading menu« bezeichnete Darstellungsform wird bislang von keinem Browser in der gewünschten Form umgesetzt.

Beispiel: **[zB]**

```
<select>
   <option>-- bitte w&auml;hlen --</option>
<optgroup label="Blumen">
   <option>Rosen</option>
   <option>Tulpen</option>
   <option>Nelken</option>
</optgroup>
<optgroup label="B&auml;ume">
   <option>Buchen</option>
   <option>Eichen</option>
   <option>L&auml;rchen</option>
</optgroup>
</select>
```

Abbildung 14.68 Auswahlgruppen in einem Auswahlfeld

14.62 option

Das `<option>`-Element erzeugt einen Eintrag in einer Auswahlliste (Selectbox oder Combo-Box), die von einem `<select>`-Container gebildet wird.

14.62.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Container
XHTML-Modul	Basic Forms, Forms
Kontext	Innerhalb des `<select>`- oder `<optgroup>`-Containers
Erlaubte Inhalte	Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 2.0, 3.2. 4.0, 4.01; XHTML 1.0, 1.1; XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<option> ... </option>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

–

14.62.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onlosecapture`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`disabled`, `label`, `selected`, `value`

14.62.3 Offizielle Attribute – W3C

disabled

Wert	*"disabled"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `disabled`-Attribut deaktiviert das betroffene `<option>`-Element. Die Option ist anschließend in der Auswahl nicht wählbar. Dies wird in der Regel durch Ausgrauen des Optionstextes dargestellt.

In XHTML ist `disabled="disabled"` zu schreiben. In HTML genügt das Einfügen des Attributnamens als Marker-Attribut.

label

Wert	*cdata*
Typ	Eine beliebige Zeichenkette
Verwendung	Optional

Barrierefreiheit: Das `label`-Attribut bestimmt den (alternativen) Namen der Option, der für den Nutzer in der Selectbox sichtbar ist. Ist kein Label gesetzt, so wird hierfür der Textinhalt des Elements verwendet. Das Attribut ist *case-sensitive*.

Anmerkung: Das `label`-Attribut von `<option>` ist für die Generierung mehrstufiger Navigationsmenüs mittels des `<select>`-Elements gedacht. Es ist analog zu dem gleichnamigen Attribut bei `<optgroup>` zu verstehen.

selected

Wert	*"selected"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `selected`-Attribut bestimmt, ob eine Option innerhalb einer Selectbox beim Laden des Formulars vorgewählt ist. Defaultzustand ist nicht vorgewählt. Innerhalb einer Selectbox ohne gesetztes `multiple`-Attribut darf nur für genau ein `<option>`-Element ein `selected`-Attribut gesetzt werden, innerhalb einer Selectbox mit gesetzten `multiple`-Attribut bei beliebig vielen Optionen.

In XHTML ist `selected="selected"` zu schreiben. In HTML genügt das Einfügen des Attributnamens als Marker-Attribut.

Anmerkung: Der Internet Explorer unterstützt auch die Werte `"true"` und `"false"`.

value

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Das `value`-Attribut bestimmt den bei Auswahl der Option an den Server übertragenen Wert. Ist kein `value`-Attribut gesetzt, so wird der Textinhalt des Elements als Wert übertragen. Das Attribut ist *case-sensitive*.

14.62.4 Erläuterung zum Element

Das `<option>`-Element erzeugt einzelne Auswahloptionen innerhalb eines mit `<select>` gebildeten Auswahlmenüs. Der Wert der gewählten Option wird zusammen mit dem `name`-Bezeichner des `<select>`-Elements verschickt. Das Element darf nur Text (`#PCDATA`) enthalten, Elementinhalt ist nicht gestattet. In

HTML ist die Endmarke als auslassbar gekennzeichnet (was in XHTML selbstverständlich nicht zulässig ist).

```
<select name="blumen">
   <option>-- bitte w&auml;hlen --</option>
   <option>Rosen</option>
   <option>Tulpen</option>
   <option>Nelken</option>
</select>
```

Abbildung 14.69 Ein einzeiliges Auswahlfeld als Dropdown-Box

Normalerweise dient der Textinhalt von `<option>` als Grundlage der bei der Auswahl verschickten Daten. Soll dies nicht geschehen, so kann alternativ ein `value`-Attribut angegeben werden:

```
<select name="blumen">
   <option value="keine Auswahl">-- bitte w&auml;hlen --</option>
   <option value="bstnr_001">Rosen</option>
   <option value="bstnr_002">Tulpen</option>
   <option value="bstnr_003">Nelken</option>
</select>
```

Ein mit dem `disabled`-Attribut gekennzeichneter Eintrag ist nicht wählbar, was durch Ausgrauung gekennzeichnet wird. Ist der erste Eintrag einer einzeiligen Auswahlliste auf diese Art deaktiviert, so ist beim Laden des Dokuments der zweite Eintrag sichtbar (allgemein: der erste, aktive Eintrag), die inaktive Option wird erst beim Aufklappen des Menüs gezeigt.

Eine Option einer Auswahlliste kann mittels des `selected`-Attributs als Defaultwert vorgewählt werden. Ist kein `multiple`-Attribut für `<select>` gewählt, so darf nur eine Option als vorgewählt gekennzeichnet werden. Sind dennoch mehrere Optionen als `"selected"` gekennzeichnet, gilt die letzte von ihnen als Vorgabewert. Ist keine Option mit `"selected"` gekennzeichnet, so wird die erste Option zum Defaultwert (sofern diese nicht deaktiviert ist).

14.63 p

Der <p>-Container bezeichnet als Blockelement einen logischen Textabsatz "paragraph". Start- und Endtag müssen stets notiert werden.

14.63.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	Text
Kontext	Im <body>-Element sowie in anderen Blockelementen mit Ausnahme von <p> selbst wie <address>, <applet>, <blockquote>, <body>, <button>, , <dd>, <del>, , <fieldset>, <form>, <iframe>, <ins>, , <map>, <noframes>, <noscript>, <object>, <td> und <th>
Erlaubte Inhalte	Inlineelemente und Text (#PCDATA)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0 (als Markerelement) HTML 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<p> ... </p>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
p  {display:block; margin: 1.33em 0; }
```

14.63.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup

Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	align
Internet Explorer	Access-, Edit-Attribute

14.63.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* (| *"justify"*)
Typ	festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte des Textabsatzcontainers `<p>`. Das Attribut steht in den strict-Vokabularen von HTML und XHTML nicht zur Verfügung.

- `align="left"` (Default)
  richtet den Inhalt des Absatzes linksbündig zum Containerrand aus.
- `align="center"`
  zentriert den Inhalt des Absatzes im Container.
- `align="right"`
  richtet den Inhalt des Absatzes rechtsbündig zum Containerrand aus.
- `align="justify"`
  richtet den Inhalt des Absatzes blocksatzmäßig aus. Dieser Wert wurde in der Vergangenheit nicht von allen Browsern unterstützt.

CSS-Alternative: Verwenden Sie das `text-align`-Property.

14.63.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.63.5 Erläuterung zum Element

Der Textabsatzcontainer `<p>` ist zur Untergliederung von Textinhalten in logische Absätze gedacht. Er besitzt dabei eine Defaultpräsentation in Form eines zum Vorgänger- und Nachfolgercontainer eingeschobenen Abstandes (die Spezifikation nennt einen `margin`-Wert von 1.33 em über und unter dem Container).

Das Element darf Inlineelemente und Text enthalten, aber keine Blockelemente. Demzufolge ist eine Verschachtelung von `<p>`-Containern (anders als bei `<div>`) nicht gestattet. Dies liegt auch daran, dass die Endmarke des Containers in HTML als auslassbar gekennzeichnet ist und die Startmarke daher stets einen neuen Absatz beginnt. In XHTML ist ein korrekter Abschluss des Elements Pflicht und sollte grundsätzlich auch in HTML erfolgen.

Auch beliebige andere Blockelemente unterbrechen einen `<p>`-Container (validierbar ist ein solcher Code natürlich nicht):

```
<!-- dies ist nicht gestattet: -->
<p>Hier beginnt ein Absatz...
<ul>
  <li>erster Eintrag</li>
  <li>zweiter Eintrag</li>
</ul>
...und das steht hinter der Liste. Hier soll der Absatz erst
enden.</p>
```

Dies tritt unangenehm in Erscheinung, wenn dem `<p>`-Container ein CSS-Style zugewiesen wurde. Aufgrund der Ergänzung der Endmarken durch den Parser wird der Container vor Beginn der Liste geschlossen. Hinter der Liste entsteht ein

Bereich, der optisch wie ein Absatz wirkt, aber lediglich einen Textbereich ohne Formatierung darstellt (für CSS ist er nicht erfassbar). Unmittelbar vor der nun überschüssigen Endmarke wird eine Startmarke ergänzt; der so komplettierte <p>-Container ist jedoch leer.

```
<!-- wird vom Parser folgendermaßen ergänzt: -->
<p>Hier beginnt ein Absatz...</p>
<ul>
  <li>erster Eintrag</li>
  <li>zweiter Eintrag</li>
</ul>
...und das steht hinter der Liste. Hier soll der Absatz erst
enden.<p></p>
```

Für den folgenden Screenshot wurde für <p> per Dokumentstil ein roter Rahmen definiert. Es ist deutlich zu sehen, dass der Container vor der Liste beendet und der Textbereich hinter der Liste nicht als Absatz gewertet wird. Der durch die Ergänzung erzeugte, leere »Geistercontainer« zeigt sich anhand seines Rahmens (siehe Abbildung 14.70).

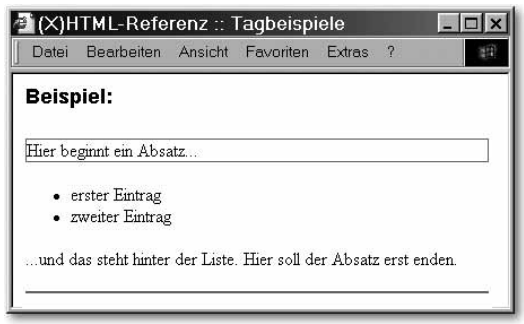

Abbildung 14.70 Unterbrechung von <p> durch ein Blockelement

Dieses Verhalten gilt für alle aktuellen Browser. Auf ein sauberes Verschachteln der Container muss daher geachtet werden.

14.64 param

Das <param>-Element dient der Definition und Übergabe von Parameterwerten an <applet>- oder <object>-Elemente, in denen es auftritt.

14.64.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Leeres Element
XHTML-Modul	Alternativ: Applet (Deprecated) oder Object
Kontext	In `<applet>`, `<object>`
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0 XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

`<param>` (HTML Syntax)

`<param />` (XHTML Syntax)

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
param  {display:none; }
```

14.64.2 Attribute (Übersicht)

Standardattribute

HTML	`id`
XHTML	`id`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

W3C	**`name`**, `type`, `value`, `valuetype`
Internet Explorer	Databinding-Attribut

14.64.3 Offizielle Attribute – W3C

name

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	**Obligatorisch**

Das `name`-Attribut gibt den Namen des Parameters an. In Abhängigkeit von der Anwendung, dem der Parameter übergeben werden soll (Java-Applet oder Multimediaobjekt), müssen entsprechende Namenskonventionen eingehalten werden, damit die Adressierung des Parameterwertes gelingt.

type

Wert	*cdata*
Typ	Ein MIME-Typ-Bezeichner
Verwendung	Optional

Das `type`-Attribut wird in Zusammenhang mit `valuetype="ref"` benötigt und nennt in diesem Falle den MIME-Typ der Parameterwertes (nach RFC 2045).

value

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Das `value`-Attribut enthält den vom `<param>`-Element an das beherbergende Objekt zu übergebenden Wert.

valuetype

Wert	*"data"* | *"ref"* | *"object"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valuetype`-Attribut legt die Art und Weise fest, in der der durch `value` bestimmte Parameterwert durch das Applet bzw. das Objekt zu interpretieren ist. Defaultwert ist `"data"`.

- `valuetype="data"` (Default)
  Das `value`-Attribut des `<param>`-Elements enthält unmittelbar den zu verwendenden Parameterwert, der daher als Zeichenkette übergeben und ausgewertet werden soll.

- `valuetype="ref"`
  Das `value`-Attribut des `<param>`-Elements enthält einen URI, der auf eine Ressource verweist, die den benötigten Wert zur Verfügung stellt. Der URI-Wert wird an das Objekt »wie er ist« übergeben, also nicht anhand eines Basis-URI aufgelöst. Das `type`-Attribut nennt, sofern erforderlich, in diesem Zusammenhang den zu erwartenden MIME-Typ des Parameterwertes.
- `valuetype="object"`
  Das `value`-Attribut enthält eine ID-Referenz auf den ID eines `<object>`-Elements im gleichen Dokument, das den Parameterwert zur Verfügung stellt.

14.64.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`dataformatas = "text" | "localized-text" | "HTML"`

`datasrc = "datenquellen-id"` (IE 4)

14.64.5 Erläuterung zum Element

Durch das leere `<param>`-Element können Informationen (benannte Parameter) an ein in die Seite eingebettetes Java-Applet oder anderes Medienobjekt weitergegeben werden, die diese zur Laufzeit benötigen. Die `<param>`-Elemente stehen hierfür im Inneren von `<applet>` oder `<object>`. Sie werden grundsätzlich nicht vom Browser dargestellt.

Zur Bezeichnung des Parameters dient das obligatorische `name`-Attribut:

```
<object>
  <param name="parametername" value="wert" />
</object>
```

Die Reihenfolge der `<param>`-Elemente innerhalb eines `<applet>`- oder `<object>`-Containers ist irrelevant, jedoch müssen sie vor eventuell vorhandenen alternativen Inhalten platziert werden. Ein Objekt wertet stets nur diejenigen Parameter aus, die für diesen Objekttyp in Frage kommen und als seine unmittelbaren Kindelemente erscheinen. Unbekannte oder nicht verwertbare Parameter werden verworfen. Es besteht auch nicht die Gefahr, dass auf die Parameter eines in einem anderen Objekt verschachteltes Alternativobjekts zugegriffen wird:

```
<object>
    <param name="parameter_aussen" value="wert" />
```

```
    <object>
        <param name="parameter_innen" value="wert" />
        <p>Alternativinhalt</p>
    </object>
</object>
```

Wird in oberem Fall das innere Objekt ausgewertet, so wird der Parameter des äußeren Objekts verworfen. Wird der Alternativinhalt benötigt, so werden alle Parameter verworfen.

14.65 plaintext

Historisch: Der `<PLAINTEXT>`-Tag stellte einen Marker dar, ab dessen Position der User Agent das Parsen des Dokuments einstellen sollte.

14.65.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Leeres Element (End-Tag inoffiziell)
XHTML-Modul	– (abgeschafft)
Kontext	Als Marker an beliebiger Position im Dokumentkörper
Erlaubte Inhalte	keine (leeres Element)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0 HTML 2.0 (deprecated), HTML 3.2 (deprecated) **abgeschafft** ab HTML 4.0
Mobil-Spezifikationen	cHTML (deprecated) *nicht enthalten* in allen weiteren Mobilspecs

Syntaxbeispiel:
`<PLAINTEXT>` [`</PLAINTEXT>`] Endtag inoffiziell

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
PLAINTEXT { display: block;
            white-space: pre;
            font-family: monospace; }
```

14.65.2 Attribute (Übersicht)

Standardattribute

HTML	–

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

W3C	–

14.65.3 Erläuterung zum Element

Der leere `<PLAINTEXT>`-Tag beendet den Parsing-Vorgang. Der User Agent interpretiert im Anschluss an die Marke also keine HTML-Tags mehr, sondern betrachtet alle folgenden Inhalte als `"text/plain"`.

Der Rest des Dokuments, der daher nicht SGML-konform sein musste, wurde als Klartext (`"plaintext"`) im Quellcode dargestellt. Dementsprechend existierte (offiziell) kein Endtag, da dieser ohnehin nicht hätte interpretiert werden können.

Gedacht war dies für längere Quellcodebeispiele, denen jedoch interpretiertes HTML vorangestellt werden sollte. Innerhalb der darzustellenden Codes sollte jedoch keine Maskierung von Steuerzeichen mehr erforderlich sein, weshalb das Parsen des Textes unterbunden wird. Als Beispiel findet man in alten Spezifikationen:

```
<PLAINTEXT>
0001 This is line one of a long listing
0002 file from <any@host.inc.com> which is sent
...
```

[zB] *Beispiel:*

Das Element wird (lustigerweise) auch in aktuellen Browsern interpretiert. Selbstverständlich sollte es nicht mehr eingesetzt werden. Beachten Sie, dass in

folgendem Screenshot (Abbildung 14.71) die Endmarken </body> und </html> dargestellt werden.

```
...
<p>Achtung, jetzt folgt <code><plaintext></code>:</p>
<plaintext>
# eine Variable wird deklariert:
$botschaft = "Hallo Welt! Dies ist PERL!";
# und ausgegeben:
print "$botschaft\n";

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

Abbildung 14.71 <plaintext> bricht den Parsing-Vorgang (Tag-Erkennung) ab.

14.66 pre

Der <pre>-Container deaktiviert die Whitespace-Normalisierung des Browsers, bewirkt also die Darstellung seines Containerinhalts unter Berücksichtigung der enthaltenen Weißraumzeichen (Leerzeichen, Zeilenumbrüche, Tabulatoren).

14.66.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	Text
Kontext	Überall in Blockelementen und Inlineelementen wie <applet>, <blockquote>, <body>, <button>, , <dd>, <del>, , <fieldset>, <form>, <iframe>, <ins>, , <noframes>, <noscript>, <object>, <td> und <th>

Erlaubte Inhalte	Inlinelemente wie <a>, <abbr>, <acronym>, <applet>, <b>, <bdo>, , <button>, <cite>, <code>, <dfn>, , <font>, <i>, <input>, <iframe>, <kbd>, <label>, <map>, <q>, <samp>, <script>, <select>, , , <textarea>, <tt> und <var> sowie Text (#PCDATA)
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<pre> ... </pre>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
pre    { display: block;
         white-space: pre;
         font-family: monospace; }
```

14.66.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	width, xml:space (nur XHTML)

Internet Explorer	Access-, Edit-Attribute
Netscape	`cols`[45], `wrap`[45]

14.66.3 Offizielle Attribute – W3C

width

Wert	*number*
Typ	Eine positive Ganzzahl
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `width`-Attribut gibt die maximale Anzahl von Zeichen pro Zeile an, ab der ein automatischer Umbruch erfolgt. Wird kein `width`-Attribut gesetzt, erfolgt kein Umbruch. Bei langen Quellcode-Zeilen zeigt der Browser in diesem Fall einen horizontalen Scrollbalken.

Anmerkung: Das Attribut ist nicht Teil der *strict*-DTDs, da es als Präsentationsattribut angesehen wird.

xml:space

Wert	*"preserve"*
Typ	Festgelegter Bezeichner
Verwendung	Optional

Das `xml:space`-Attribut darf in XHTML-Dokumenten verwendet werden. Es sollte dabei sinnigerweise nur mir dem Wert `"preserve"` eingesetzt werden.

14.66.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"` (IE 5)

`unselectable = "on" | "off"` (IE5)

45 Netscape 4; später abgeschafft.

14.66.5 Proprietäre Attribute – Netscape

cols

Wert	*number*
Typ	Eine positive Ganzzahl
Verwendung	Optional

(Historisch) Das `cols`-Attribut gibt in Netscape 4 analog zum offiziellen `width`-Attribut die maximale Anzahl von Zeichen pro Zeile an, ab der ein automatischer Umbruch erfolgt. Gleichzeitig wird der Zeilenumbruch grundsätzlich gestattet, als ob explizit das Netscape-Attribut `wrap` gesetzt wäre.

wrap

Wert	–
Typ	Leeres Marker-Attribut
Verwendung	Optional

(Historisch) Das `wrap`-Attribut erlaubt in Netscape 4 den automatischen Zeilenumbruch innerhalb des `<pre>`-Elements, wenn die Zeile zu breit für den Darstellungsbereich des Browsers wird.

14.66.6 Erläuterung zum Element

Das Element `<pre>` deaktiviert für das Innere seines Containers das reguläre Whitespace-Verhalten des Browsers, der ansonsten Leerzeichen, Tabulatoren und Zeilenumbruchzeichen normalisiert. Innerhalb von `<pre>` werden sämtliche Whitespace-Zeichen vom Parser an die Formatiererstufe weitergereicht, also im Browserfenster dargestellt.

Somit ist es möglich, die Ausgabe im Quelltext vorzuformatieren (`"preformatted"`), beispielsweise zur Tabellierung von Inhalten (auch die früher beliebten ASCII-Bilder können so erzeugt werden):

```
<pre>
                Tschiep!
                   \__     .-.
                         /'V'\
                        (/   \)
                       ==="="===
                          /_\
</pre>
```

Der Containerinhalt wird gewöhnlich in einer Serifenschrift konstanter Laufweite (meist Courier oder Courier New) präsentiert. Im Element dürfen nur Text- oder Inlineelemente erscheinen, keine Blockelemente.

Als Inhalte innerhalb des Containers darüber hinaus für alle DTD-Versionen *verboten* sind die Elemente `<img>`, `<object>`, `<big>`, `<small>`, `<sub>` und `<sup>`, also alle Elemente, die potentiell auf die Zeilenhöhe Einfluss nehmen. In der *transitional*-DTD sind daher zusätzlich `<applet>`, `<font>` und `<basefont>` ausgeschlossen. In den HTML-DTDs ist dies über Ausschlussbedingungen formuliert. In den XHTML-DTDs ist dies nicht direkt möglich, das Inhaltsmodell entspricht jedoch dem von HTML.

Beispiel: **[zB]**

```
<pre>Das ist PRE-Text,
   hier mit Leerzeichen.
        hier mit Tabulatoren
                bla.
        wieder Leerzeichen.
</pre>
```

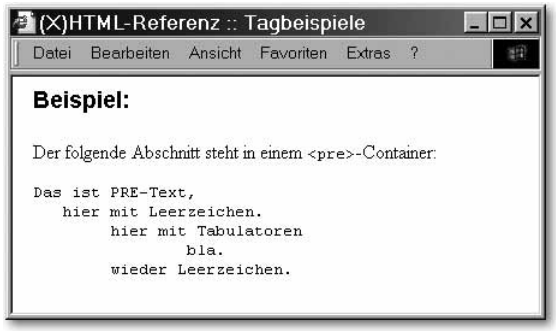

Abbildung 14.72 Vorformatierter Quelltext mit <pre> dargestellt

14.67 q

Logische Auszeichnung: Das Element `<q>` zeichnet seinen Inhalt als Zitat (*quotation*) mit einer Quellenangabe aus und fügt Anführungszeichen hinzu.

14.67.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Überall in Block und Inlineelementen
Erlaubte Inhalte	Inlineelemente (auch `<q>` selbst) und Text (`#PCDATA`)
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1; XHTML Basic **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile *nicht enthalten* in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<q> ... </q>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
q:before {content: open-quote }
q:after  {content: close-quote }
```

14.67.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onblur`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onkeydown`, `onkeypress`, `onkeyup`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`cite`
Internet Explorer	Access-, Edit-Attribute

14.67.3 Offizielle Attribute – W3C

cite

Wert	*URL*
Typ	ein URI-String
Verwendung	Optional

Das `cite`-Attribut enthält eine Quellenangabe des Zitats in Form eines URL, der auf die zitierte Ressource zeigt. Die häufig zu findende Praxis einer Klartextquellenangabe ist streng genommen falsch.

14.67.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.67.5 Erläuterung zum Element

Das Element kennzeichnet ein Inlinezitat, ist also ein Pendant zum Blockelement `<blockquote>`, das dementsprechend Blockzitate markiert. Als Inhalt des `<q>`-Containers sind nur Inlineelemente und Text zugelassen. Die Zeilenumbruchmarke `<br>` ist allerdings gestattet, ebenfalls eine Verschachtelung von `<q>` mit sich selbst.

Zur Präsentation des eingeschlossenen Textes als Zitat werden vor und hinter diesem durch den Parser Anführungszeichen eingefügt. Zusätzlich kann über das optionale `cite`-Attribut die Zitatquelle genannt werden (dies soll, so die ursprüngliche Intention des Attributs, in Form eines URI geschehen). Das `lang`- bzw. `xml:lang`-Attribut kann in diesem Zusammenhang dazu dienen, die Sprache des Zitats zu kennzeichnen.

[»] *Hinweis:* Das Element <q> wird im Internet Explorer bis einschließlich Version 7.0 nicht unterstützt. Es werden im Internet Explorer folglich auch keine Anführungszeichen eingeblendet.

Steuerung der Anführungszeichen mit CSS

Mittels des quotes-Properties kann (zumindest in Browsern, die dies unterstützen) die Art der einzusetzenden Anführungszeichen auch für verschachtelte <q>-Container festgelegt werden.

Hier werden die amerikanischen Anführungszeichen angeordnet – die doppelten für die äußere und die einfachen für die verschachtelte Zitatebene.

```
q    { quotes: '"' '"' "'" "'"; }
```

Analog könnten deutsche Anführungszeichen angegeben werden:

```
q    { quotes: '»' '"' "," "‘"; }
```

Wollen Sie die Ausgabe von Anführungszeichen um den Containerinhalt unterdrücken, so setzen Sie einfach:

```
q    { quotes: none; }
```

Sollen sprachspezifische Anführungszeichen gesetzt werden, kann man entweder mit einer entsprechenden Styleklasse arbeiten oder die Pseudoklasse :lang verwenden, die auf den Wert des lang-Attributs Bezug nimmt (das Attribut muss in diesem Falle gesetzt sein):

```
q.franz    { quotes: '«' '»' "<" ">"; }

q:lang(fr) { quotes: '«' '»' "<" ">"; }
```

Anmerkung: Es ist von Vorteil, den Charactercode des Dokuments auf ISO-8859–1 zu setzen, wenn Sie andere als die hochgestellten amerikanischen Anführungszeichen einsetzen möchten. Verwenden Sie hierfür das entsprechende <meta>-Element.

[zB] *Beispiel:*

```
<p>Ein Inlinezitat:</p>

<p>Hamlet sagte: <q>Sein oder nicht sein.</q></p>

<!-- cite nicht korrekt eingesetzt: -->
<p>Hamlet sagte:
<q cite="Shakespeare, Hamlet, 3.Akt, 1. Szene">
Sein oder nicht sein; das ist hier die Frage:<br>
Obs edler im Gem&uuml;t... </q></p>
```

```
<!-- cite korrekt eingesetzt: -->
<p>Das W3C sagt &uuml;ber das <code>cite</code>-Attribut: <q
cite="http://www.w3.org/TR/html401/struct/text.html#edef-Q">The
value of this attribute is a URI that designates a source document
or message. This attribute is intended to give information about the
source from which the quotation was borrowed.</q></p>
```

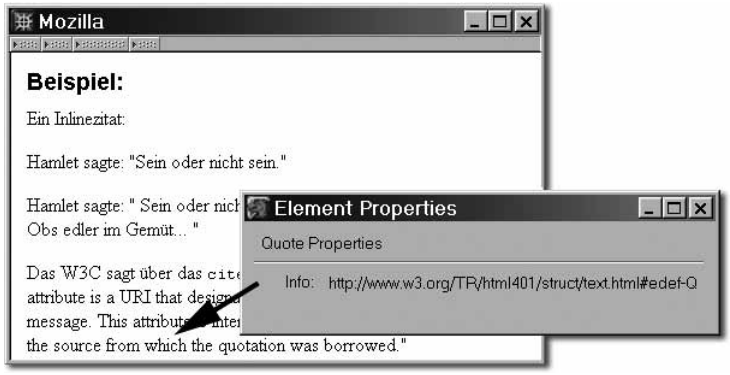

Abbildung 14.73 Mozilla zeigt den URL des cite-Attributs über den Befehl Properties des Kontextmenüs.

14.68 s

Präsentationselement: Das Element <s> (*strikethrough*) präsentiert den eingeschlossenen Inhalt durchgestrichen.

14.68.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Legacy
Kontext	überall in Block und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0/4.01 (tr/fs), XHTML 1.0 (tr/fs) **abgeschafft** in HTML 4.0/4.01 strict XHTML 1.0 strict, 1.1, 2.0
Mobil-Spezifikationen	– (nirgends spezifiziert)

Syntaxbeispiel:

```
<s> ... </s>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
s       { display: inline;
          text-decoration: line-through; }
```

14.68.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.68.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.68.4 Erläuterung zum Element

Das Element dient der Präsentation einer Textpassage als durchgestrichen, besitzt dabei jedoch nicht streng die logische Bedeutung »gelöscht« oder »gestrichen«. Für diesen Zweck verwenden Sie alternativ das Element `<del>`. Als reines Präsentationselement steht `<s>` genau wie das synonyme Element `<strike>`, auf der W3C-Liste für abzuschaffende Elemente und gilt als »deprecated«.

Beispiel: **[zB]**

```
<p><s>Dieser Text ist gestrichen.</s></p>
```

Abbildung 14.74 Der <s>-Container präsentiert seinen Inhalt durchgestrichen.

14.69 samp

Logische Auszeichnung: Das Element `<samp>` markiert seinen Elementinhalt als »Beispiel« (*sample*) – meist im Sinne eines kurzen Quelltextbeispiels einer Programmiersprache.

14.69.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlinelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile *nicht enthalten* in iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<samp> ... </samp>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
samp     { display: inline;
           font-family: monospace; }
```

14.69.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.69.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.69.4 Erläuterung zum Element

Das Element <samp> gehört zur Gruppe der »Phrase«-Elemente. Es dient der Markierung eines kurzen Beispiels (*sample*), das aus einem Programmquelltext stammen kann. Der Containerinhalt wird gewöhnlich in einer Serifenschrift konstanter Laufweite (meist Courier oder Courier New) präsentiert. Im Element dürfen nur Text- oder Inlineelemente erscheinen, keine Blockelemente.

Inhaltlich ist das Element verwandt mit <code>, wobei dies explizit den Inhalt als Programmcode kennzeichnet. Für mehrzeilige Beispiele in <samp> müssen Sie Zeilenumbruchmarken <br/> setzen. Alternativ können Sie den Container <pre> verwenden, der den Conteinerinhalt nicht normalisiert, sondern die Umbrüche an den Browser weitergibt.

Beispiel: **[zB]**

```
<p>
<samp>Quelltextbeispiel als sogenanntes "Sample"</samp>
</p>

<p><strong>Zeilenumbr&uuml;che werden dabei nicht
beachtet:</strong><br>
<samp>
# eine Variable wird deklariert:
$botschaft = "Hallo Welt! Dies ist PERL!";
# und ausgegeben:
print "$botschaft\n";
</samp></p>

<p><font face="Comic Sans MS" size="3">Es darf allerdings nur
<samp>Inlinetext</samp> enthalten sein, keine
Blockelemente.</font></p>

<p><samp>So wie hier:
<strong> Viel Spa&szlig; dabei!</strong>
</samp></p>
```

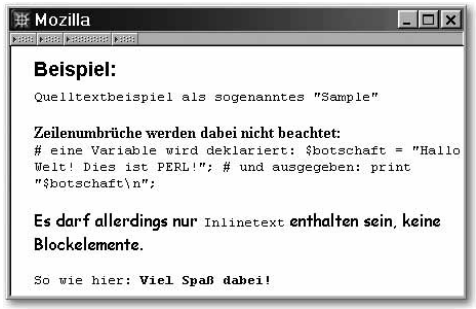

Abbildung 14.75 Das <samp> stellt seinen Inhalt in Monospace-Font dar.

14.70 script

Das Element `<script>` kennzeichnet seinen Inhalt als Quellcode in einer clientseitig interpretierten Scriptsprache, deaktiviert für den Containerbereich die HTML-Interpretation und startet den benötigten Scriptinterpreter.

14.70.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Container, unsichtbar
XHTML-Modul	Scripting
Kontext	im Dokumentkopf `<head>`, überall im `<body>` in Block- und Inlineelementen
Erlaubte Inhalte	`#CDATA` (Scriptcode) in HTML `#PCDATA` (Scriptcode) in XHTML
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01; XHTML 1.0, 1.1 **abgeschafft** in XHTML 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<script type="text/javascript"> ... </script>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

– (keine; Element wird nicht dargestellt)

14.70.2 Attribute (Übersicht)

Standardattribute

HTML	–
XHTML	–

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–

Internet Explorer Events	`onerror`, `onload`, `onpropertychange`, `onreadystatechange`

Elementspezifische Attribute

W3C	`charset`, `defer`, `language`, `src`, **`type`**
Internet Explorer	`event`, `id`, `for`, `runat`

14.70.3 Offizielle Attribute – W3C

charset

Wert	*encoding-bezeichner*
Typ	Ein registrierter Bezeichner für Character-Encodings[46].
Verwendung	Optional

Das `charset`-Attribut wird in Zusammenhang mit externen, über das `scr`-Attribut eingebundenen Scripten eingesetzt. Es teilt dem Browser das zu erwartende Encoding-Schema der Ressource mit. Default-Encoding ist ISO 8859–1 (ISO Latin 1), eine Untergruppe von ISO 8859[47].

defer

Wert	*"true"* | *"false"*
Typ	ein Boolescher Wert
Verwendung	Optional

Das `defer`-Attribut teilt dem User Agent mit, dass ein Scriptblock nicht unmittelbar Output im Dokument erzeugt (z. B. per `document.write()`), seine Auswertung also ohne Nebenwirkungen aufgeschoben (»*deferred*«) werden kann.

- `defer="false"` (Default)
  Der Scriptblock soll unmittelbar ausgewertet werden.
- `defer="true"`
  Der Scriptblock muss nicht unmittelbar ausgewertet werden.

Ein User Agent kann entsprechend den Scriptblock (vorläufig) überspringen und mit dem Parsing des Dokuments fortfahren, um so den Renderingvorgang zu beschleunigen.

46 Siehe http://www.iana.org/assignments/character-sets.

47 Mehr darüber bei »ISO 8859 Alphabet Soup« (Roman Czyborra): www.unicodecharacter.com/charsets/iso8859.html.

language

Wert	*cdata*
Typ	ein Sprachbezeichner mit optionalem Versionsstand
Verwendung	Optional
Gültigkeit	**abgeschafft** in HTML 4.01 Strict, XHTML 1.0 Strict, XHTML 1.1.

Deprecated: Das `language`-Attribut nennt Namen und Versionsstand der Scriptsprache des `<script>`-Containers. Einen Standard für die Namensnennung festzulegen, ist nicht gelungen, weshalb das W3C dieses Attribut ablehnt.

```
<!-- Script-Container enthält explizit JavaScript 1.2: -->
<script language="JavaScript 1.2"> ... </script>
```

[»] *Hinweis:* Das `language`-Attribut ist veraltet und sollte zugunsten des obligatorischen und inhaltlich gleichwertigen `type`-Attributs vermieden werden.

src

Wert	*URL*
Typ	ein URI-String, der die externe Scriptdatei bezeichnet
Verwendung	Optional

Das `src`-Attribut nennt den URI einer externen Datei, welche die in Zusammenhang mit dem `<script>`-Container auszuführenden Scriptdaten enthält. Die externen Scriptdaten verhalten sich, als ob sie direkt im Container enthalten wären.

Der Sprachtyp wird dem Browser über das `type`-Attribut mitgeteilt. Ansonsten gibt auch die Dateiendung Auskunft über den Sprachtyp, wobei der Suffix `.js` eine externe JavaScript-Datei mit dem MIME-Typ `"application/x-javascript"` kennzeichnet.

[»] *Hinweis:* Innerhalb von XHTML-Dokumenten bietet sich die Verwendung von externen Scriptdateien an, um das umständliche Umgeben des Scriptblocks mit `<![CDATA[`-Kennungen zu vermeiden.

type

Wert	*content-type*
Typ	ein Content-Typ-Bezeichner
Verwendung	**Obligatorisch**

Das obligatorische `type`-Attribut nennt den Inhaltstyp des Script-Containers, mit anderen Worten die Scriptsprache. Es existiert kein Defaultwert. Das Attribut ist nicht *case-sensitive*.

Für JavaScript ist der Wert `"text/javascript"` zu übergeben:

```
<script type="text/javascript"> . . . </script>
```

14.70.4 Proprietäre Attribute – Internet Explorer

event

Wert	*cdata*
Typ	ein Event-Typ-Bezeichner
Verwendung	Optional

Das `event`-Attribut ordnet einem Scriptblock im Internet Explorer einen Eventtyp zu, mit dem im Zusammenhang der Scriptblock ausgeführt werden soll. Der genannte Event muss dabei an einem Objekt auftreten, das durch den `IDREF` des `for`-Attributs des Scriptblocks bezeichnet ist.

Hier wird ein Scriptblock an den Klick-Event eines Buttons gebunden, ohne dass im Button ein Eventhandler erforderlich ist:

```
<script for="mein_button" event="onclick()">... </script>

<button id="mein_button">Hier Klicken</button>
```

id

Wert	*ID*
Typ	ein Bezeichner vom Typ `ID`
Verwendung	Optional

Das `id`-Attribut dient im Internet Explorer der Benennung eines Scriptblocks. Dies kann benötigt werden, wenn ein Scriptblock mit XML-Dateninhalt als Ersatz für eine XML-Dateninsel eingesetzt wird, um per IDREF (oder per `document.all`) auf dessen Inhalt zugreifen zu können.

```
<script language="xml" id="xml1">
...  xml-daten
</script>
```

Anmerkung: Das Standardattribut `id` ist für `<script>` nicht offiziell definiert.

for

Wert	*idref*
Typ	ein Bezeichner von Typ ID
Verwendung	Optional

Das `for`-Attribut nennt für den Internet Explorer in Form einer ID-Referenz, den ID des Objekts, an den ein Scriptblock gebunden werden soll. Gleichzeitig wird mittels des `event`-Attributs festgelegt, welcher Event am beobachteten Objekt die Ausführung des Scriptblocks auslöst.

Hier wird ein Scriptblock mittels des `for`-Attributs an einen Button gebunden. Das Script wird bei Auftreten des Klick-Events gestartet:

```
<script for="mein_button" event="onclick()"> ... </script>

<button id="mein_button">Hier Klicken</button>
```

runat

Wert	*"server"*
Typ	festgelegter Bezeichner
Verwendung	Obligatorisch für serverseitige ASP-Scripte

Das `runat`-Attribut bestimmt mit `runat="server"`, ob das Script als ASP-Script (Active Server Pages) serverseitig ausgeführt werden soll. Es richtet sich speziell an Microsoft-Server und ist für serverseitige Codeblöcke obligatorisch. (Ein etwaiger korrespondierender Wert `runat="client"` macht wenig Sinn. Für clientseitig ausgeführte Scripte bräuchte er nicht explizit gesetzt werden. Die Dokumentation bei Microsoft, ob dieser Wert real existiert, ist uneindeutig.)

```
<script language="C#" runat="server">
   /*  hier serverseitig interpretierter C#-Code */
</script>
```

[»] *Hinweis:* Es darf gleichzeitig das `src`-Attribut verwendet werden, um den Server anzuweisen, eine externe Scriptdatei zur Verarbeitung einzubinden.

14.70.5 Erläuterung zum Element

Um Bereiche mit clientseitig interpretierten Scriptsprachen in HTML-Dokumente einfügen zu können, wurde in HTML 3.2 der `<script>`-Container eingeführt. Er darf sowohl im `<head>` als auch im `<body>` des Dokuments in beliebiger Anzahl auftreten, wobei die Blöcke in der Reihenfolge abgearbeitet werden, in der sie im Dokument erscheinen (Scripts im Head also zuerst).

Da prinzipiell eine Vielzahl von Scriptsprachen verwendet werden dürfen, muss der MIME-Typ der jeweiligen Sprache über das Pflicht-Attribut `type` angekündigt werden. Das ursprünglich in gleicher Bedeutung verwendete `language`-Attribut ist veraltet und sollte nicht länger eingesetzt werden – eine Folge daraus ist, dass nur noch die Sprache, aber nicht mehr ihr Versionsstand angegeben werden kann (was de facto keinen Verlust darstellt).

Ein Scriptblock sieht daher minimal wie folgt aus:

```
<script type="text/javascript">

   // hier der auszuführende Quellcode in JavaScript

</script>
```

Inhaltsmodell

Der Inhaltsbereich des Scriptblocks ist in **HTML** als `CDATA` definiert, enthält für den Browser also reine Zeichendaten, die nicht geparst werden. Aus diesem Grund dürfen im Containerinneren die Tagklammern »<« und »>« ohne Maskierung verwendet werden. Es werden im Containerinneren also keine HTML-Tags erkannt und auch keine Entities aufgelöst. Positiv zu vermerken ist, dass kein Gedanke daran verschwendet werden muss, ob im Programmquellcode erscheinende Zeichen besondere Bedeutung besitzen.

In **XHTML** ist `CDATA` als Inhaltsmodell für Elemente nicht möglich (die Syntax für XML-DTDs enthält diese Option nicht), weswegen in diesem Fall das Inhaltsmodell `(#PCDATA)` gilt. Das Containerinnere wird also geparst, weshalb die Steuerzeichen »<«, »>« und »&« mit den Entities `<`, `>` und `&` maskiert werden müssen. Hier besteht ein Problem der Abwärtskompatibilität zu `<script>` in HTML: Da dort der Bereich *nicht geparst* wird, können diese Entities auch *nicht ersetzt* werden. Die Notwendigkeit einer solchen Maskierung muss daher auch in XHTML unbedingt umgangen werden.

Maskierung des Containerinhaltes in HTML

Für Scriptcontainer innerhalb von HTML-Dokumenten bietet sich ein Auskommentieren des gesamten Inhaltes mittels HTML-Kommentaren an. Dies richtet sich nicht an scriptfähige Browser (im `CDATA`-Inhaltsmodell werden Kommentare nicht erkannt), sondern an jene User Agents, die den `<script>`-Tag nicht kennen und daran gehindert werden müssen, den Inhalt als Text darzustellen. Ein so auskommentierter Scriptblock besitzt die Form:

```
<script type="text/javascript">
<!-- für alte Browser auskommentiert
```

```
   // hier folgt JavaScript

// Ende der Auskommentierung -->
</script>
```

Ältere Browser sehen nur einen Kommentar und entfernen diesen. Ein scriptfähiger Browser ignoriert die Kommentarzeile am Anfang (beginnen Sie mit dem eigentlichen Script also nie in dieser Zeile!) und schaltet in den Scriptmodus. Bis zum Auftreten der Endmarke `</script>` erfolgt kein Parsing. Um das Ende des HTML-Kommentars vor dem Scriptinterpreter zu verstecken, wird dieses wiederum gemäß JavaScript-Kommentarsyntax auskommentiert.

Problem mit der Zeichenfolge »</« in Scripten

Trotz des deaktivierten Parsings muss der Browser die Endmarke des Containers erkennen können. Hierzu achtet er auf die Zeichenfolge »</«, was dazu führen kann, dass der Beginn einer per `document.write()` einzufügenden HTML-Endmarke bereits den Scriptblock beendet, was zu einem Fehler führt:

```
// problematisch:
document.write('<p>Ich f&uuml;ge einen Absatz ein.</p>');
```

Maskieren Sie daher den Slash der Endmarke mit *Backslash*:

```
// unproblematisch:
document.write('<p>Ich f&uuml;ge einen Absatz ein.<\/p>');
```

Diese Methode ist grundsätzlich anzuraten – unabhängig davon, ob das Script im HTML- oder im XHTML-Kontext auftritt.

Maskierung des Containerinhaltes in XHTML

In XHTML besitzt der Script-Container das Inhaltsmodell `(#PCDATA)`. Dies bedeutet (abgesehen von der Notwendigkeit der Maskierung entsprechender Zeichen) auch, dass Kommentare erkannt und beachtet werden. Eine Auskommentierung des Scriptblocks wie von HTML her gewohnt führt dazu, dass für einen ordnungsgemäß arbeitenden Parser der Kommentarinhalt und damit das Script »verschwindet«. Auf die Auskommentierung *muss* daher in XHTML verzichtet werden.

Eine derartige Auskommentierung ist insbesondere in XHTML 1.1 gefährlich, da solche Dokumente als `application/xhtml+xml` ausgeliefert werden *sollen*, vom Client also mit Sicherheit als XML behandelt werden. Liefert man hingegen XHTML 1.0 weiterhin als `text/html` aus (weitverbreitete, halbwegs zu rechtfertigende Praxis), mag man damit durchkommen.

Weiterhin soll zwecks Abwärtskompatibilität zu HTML keine Maskierung von Zeichen durch Entities erfolgen müssen (da diese in HTML nicht expandiert würden). Zu diesem Zweck kann in XML ein so genannter CDATA-Bereich eingefügt werden, quasi ein SGML-Befehl, der das Parsing für den eingeschlossenen Bereich deaktiviert:

```
<![CDATA[  dies ist CDATA-Inhalt  ]]>
```

An den Inhalt wird als einzige Bedingung gestellt, dass die Zeichenfolge »]]>«, die den Bereich beendet, im Inneren nicht verwendet werden darf.

Mit definiertem CDATA-Bereich kann ein Script in XHTML ohne weitere Einschränkung in der gleichen Form bleiben, wie für HTML erforderlich:

```
<script type="text/javascript">
<![CDATA[

   // hier darf jetzt <, > und & eingesetzt werden

]]>
</script>
```

Wird dies nun von einem nicht XHTML-fähigen User Agent verarbeitet, so können die CDATA-Marken Probleme verursachen. Man könnte sie zunächst vor den Scriptinterpreter durch JavaScript-Kommentare verstecken:

```
<script type="text/javascript">
// <![CDATA[

   // hier darf jetzt <, > und & eingesetzt werden

// ]]>
</script>
```

Um auch noch ältere User Agents einzubeziehen, muss auch noch der HTML-Kommentar eingeflochten werden. Die hierdurch erforderliche Zeichenfolge nimmt allerdings langsam lächerliche Proportionen an:

```
<script type="text/javascript">
   <!--//--><![CDATA[//><!--

   // hier darf jetzt <, > und & eingesetzt werden

   //--><!]]>
</script>
```

Salomonische Lösung: Externe Scriptdateien

Um die eben angeschnittene Problematik, die sich aus den Unterschieden der Inhaltsmodelle von `<script>` zwischen HTML und XHTML ergeben, zu umschiffen, besteht die einfachste Lösung darin, überhaupt keine Inhalte mehr im Scriptcontainer zuzulassen und stattdessen auf das Einbinden extern gelagerter Scriptdateien auszuweichen. Dies geschieht mit dem `src`-Attribut von `<script>`.

Der `<script>`-Container muss weiterhin geschlossen werden, wobei, obwohl er leer bleibt, *keine* Verkürzung zu `<script ... />` gestattet ist. Da der Container nun keinen Inhalt mehr besitzt (genauer: in diesem Fall keinen besitzen darf!), braucht weder ein Kommentar noch ein `CDATA`-Bereich eingesetzt werden. Nach wie vor Pflicht ist das `type`-Attribut:

```
<script type="text/javascript" src="extern.js"></script>
```

Ein weiterer Vorteil der Externalisierung besteht in der Möglichkeit, ein Script mehrfach einzusetzen. Vor allem die Einbindung von Programmbibliotheken wie Prototype oder Scriptaculous erfolgt üblicherweise auf diesem Wege.

Praxistipp: Externalisieren Sie Ihre Scripte

Binden Sie Scriptdaten besser grundsätzlich per `src`-Attribut als externe Dateien ein. Sie ersparen sich zukünftige Scherereien. Die Lösung ist obendrein auch zu älteren Browsern grundsätzlich abwärtskompatibel.

Kennzeichnung des MIME-Typs von Inlinescripten

Der MIME-Typ eines Inlinescripts, wie es innerhalb von Eventhandler-Attributen auftritt, kann nicht unmittelbar mitgeteilt werden. Defaultmäßig wird der Browser von JavaScript ausgehen, eine Eindeutigkeit besteht jedoch nicht:

```
<!-- welche Scriptsprache wird hier eingesetzt? -->
<p><a href="#" onclick="alert('Sie haben
 geklickt!')">Klicken Sie hier mal.</a></p>
```

Die Lösung besteht darin, globale eine Default-Scriptsprache zu definieren, die für alle Inline-Scriptanweisungen gilt. Dies geschieht mittels `<meta>`-Tag:

```
<meta http-equiv="Content-Script-Type"
 content="text/javascript"/>
```

[»] *Hinweis:* Ein analoges Vorgehen ist für über das `style`-Attribut zugewiesene Inlinestyles möglich. Mehr über das `<meta>`-Element im entsprechenden Abschnitt.

XHTML und Scripting

Eine Grundeigenschaft eines XHTML-Dokuments und der eigentliche Vorzug gegenüber HTML-Dokumenten besteht in seiner **Wohlgeformtheit** (sofern das Dokument korrekt geschrieben ist). Diese bildet die Grundlage dafür, dass das Dokument in einem XML-Kontext erfolgreich verarbeitet werden kann, beispielsweise indem Informationen aus dem Dokument extrahiert werden.

Der Inhalt oder der Aufbau des Dokuments kann jedoch durch Scripting verändert werden. Eine »alte Gewohnheit« von Webprogrammierern besteht darin, dies mit Hilfe der Funktion `document.write()` zu tun. Hiervon muss in Zusammenhang mit XHTML *abgeraten* werden, da hierbei (zumindest potentiell) die Wohlgeformtheit des Dokuments gefährdet ist.

Verwenden Sie stattdessen die Methoden wie `appendChild()`, `setAttribute()` und andere, die das DOM zur Verfügung stellt. Eine weitere Diskussion dieses Aspekts ist an dieser Stelle nicht möglich – ich möchte Sie stattdessen auf entsprechende Fachliteratur verweisen (beispielsweise Christian Wenz, »JavaScript und AJAX«, Galileo Computing).

Praxistipp: Manipulation des Inhalts von XHTML-Dokumenten

Verwenden Sie in XHTML-Dokumenten kein `document.write()`, sondern die DOM-Funktionen zur Manipulation des Dokumentbaums. Die Wohlgeformtheit des Dokuments bleibt so gewährleistet.

Barrierefreiheit

Es müsse darauf geachtet werden, dass die Seite ihren vollen Informationswert behält, selbst wenn Scripting nicht unterstützt wird oder absichtlich deaktiviert ist, so die Aussage von WAI, Abschnitt 6.3, bzw. BITV, Abschnitt 7.3. Sie sollten also, falls Informationen über Scripte eingefügt werden, einen `<noscript>`-Container vorsehen, der zumindest das mögliche Fehlen von Information anzeigt.

14.71 select

`select` dient innerhalb von Formularen als Container für Optionselemente zum Erzeugen von Selectboxen (Combo-Boxen) oder Auswahlfeldern.

14.71.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Basic Forms, Forms
Kontext	Innerhalb eines `<form>`-Containers in Block- oder Inlineelementen
Erlaubte Inhalte	`<option>`-Elemente in beliebiger Anzahl
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1; XHTML Basic **abgeschafft in:** XHTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<select> ... </select>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
select      { display:inline; }
```

14.71.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	`onblur`, `onchange`, `onfocus`
Internet Explorer Events	`onafterupdate`, `onbeforeupdate`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onerrorupdate`, `onhelp`, `onlosecapture`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onscroll`, `onselectstart`

Elementspezifische Attribute

W3C	`disabled`, `multiple`, `name`, `size`, `tabindex`
Internet Explorer	Access-, Databinding-, Edit-Attribute

14.71.3 Offizielle Attribute – W3C

disabled

Wert	*"disabled"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `disabled`-Attribut deaktiviert ein Select-Element und damit die Auswahl einer seiner Optionen (Zustandsänderung). Gleichzeitig wird die Versendung eines Werts bei Abschicken der Formulardaten unterdrückt, auch wenn eine Option vorgewählt sein sollte. Ein deaktiviertes Select-Element wird durch Ausgrauung präsentiert.

Anmerkung: Der Internet Explorer unterstützt auch die Werte `"true"` (Feld deaktiviert) und `"false"` (nicht deaktiviert; Default).

multiple

Wert	*"multiple"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `multiple`-Attribut dient dazu, für eine Selectbox die Auswahl mehrerer Optionen gleichzeitig zu gestatten (eine zahlenmäßige Beschränkung ist dabei nicht möglich). Der Defaultzustand ohne gesetztes `multiple`-Attribut erlaubt nur die Wahl einer einzigen Option. Ist `multiple` gesetzt, so werden beim Datenversand mehrere Werte entsprechend der Anzahl der gesetzten Optionen übertragen.

Anmerkung: Der Internet Explorer unterstützt auch die Werte `"true"` (Mehrfachwahl gestattet) und `"false"` (nur eine Option wählbar; Default).

name

Wert	*nmtoken*
Typ	Ein Bezeichner in Form eines NMTOKEN
Verwendung	Optional

Das `name`-Attribut legt den Bezeichner (*control name*) der Selectbox innerhalb eines Formulars fest. Der Bezeichner findet bei der Datenübertragung als name-Part eines »name-value«-Pärchens Verwendung, wobei sich der in Zusammenhang mit der Selectbox übertragene Wert durch den der ausgewählten Option(en) bestimmt (die `<option>`-Elemente besitzen keine eigenen Namen).

[»] *Hinweis:* Der Attributwert ist nicht *case-sensitive*, daher werden beispielsweise `name="FELDNAME"` und `name="feldname"` als identisch betrachtet.

size

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Optional

Das `size`-Attribut bestimmt die Anzahl der beim Rendern der Selectbox im Browser gleichzeitig sichtbaren Optionseinträge, mithin die Höhe der Selectbox in Zeilen. Der Defaultwert ohne `size`-Attribut ist `"1"`.

tabindex

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Optional

Barrierefreiheit: Die an das Attribut `tabindex` übergebene Ganzzahl legt, in aufsteigender Richtung, die Position des Select-Elements in der Tabulatorwahl-Reihenfolge (⇆-Taste) fest.

Die Tab-Reihenfolge gestaltet sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes `tabindex`-Attribut in Quelltextreihenfolge. (Zu den aktivierbaren Elementen gehören allgemein neben `<select>` auch `<a>`, `<area>`, `<button>`, `<input>`, `<object>` und `<textarea>`.)

[»] *Hinweis:* Die Tab-Reihenfolge enthält alle aktivierbaren Elemente gemeinsam, also auch Links. Für Links und Formularelemente sollten daher niemals gleiche Tabindex-Werte gesetzt werden.

14.71.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.71.5 Erläuterung zum Element

Das `<select>`-Element dient dazu, innerhalb eines Formulars eine Reihe von Auswahlmöglichkeiten in Form eines Optionsfeldes darzustellen. Dies kann entweder einzeilig sein (genau eine Option ist sichtbar) oder mehrzeilig (zwei oder mehr Optionen gleichzeitig sichtbar). Ebenso können eine oder mehrere der vorliegenden Optionen als gleichzeitig anwählbar definiert sein.

Als Inhalt von `<select>` sind `<option>`- und `<optgroup>`-Container erlaubt. Letztere dienen dazu, eine Reihe von Optionen zu einer Gruppe zusammenzufassen. Mindestens eines der beiden Elemente muss vorkommen, d.h. der `<select>`-Container darf nicht leer sein.

Datenversand

Um den Datenversand zu ermöglichen, ist über das `name`-Attribut ein *control name* für `<select>` zu setzen. Übertragen wird der Wert der gewählten Option.

```
<select name="das_selectfeld">
   <!-- eine Option wählbar -->
</select>
```

Mehrfachauswahl und Mehrzeiligkeit

Ein Select-Feld in der oben angegebenen Form erzeugt eine einzeilige Dropdown-Box (Combo-Box) bei dem maximal eine Option angewählt werden kann. Sollen zwei oder mehr Optionen gleichzeitig wählbar sein, so muss das `multiple`-Attribut gesetzt werden. In HTML ist dies ein Marker-Attribut ohne Wert-

zuweisung, in XHTML muss die Form `multiple="multiple"` verwendet werden. Hiermit ist eine Mehrfachauswahl möglich:

```
<select name="das_selectfeld" multiple="multiple">
   <!-- mehrere Optionen wählbar -->
</select>
```

Da die Auswahl mehrerer Optionen bei einzeiligen Dropdown-Boxen wenig intuitiv ist, empfiehlt es sich, stattdessen eine mehrzeilige Variante vorzuziehen. Mittels des `size`-Attributs wird bestimmt, wie viele Optionen gleichzeitig sichtbar sein sollen. Dies erzeugt ein mehrzeiliges Auswahlfeld:

```
<select name="das_selectfeld" multiple="multiple" size="5">
   <!-- fünf Optionen sichtbar, mehrere wählbar -->
</select>
```

Übersteigt die Anzahl der Optionen im Containerinneren die Anzahl der durch `size` sichtbar gemachten Zeilen, so wird ein Scrollbalken gezeigt.

[zB] *Beispiel:*

```
<select name="gartendekoration">
   <option>-- bitte w&auml;hlen --</option>
<optgroup label="Blumen">
   <option>Rosen</option>
   <option>Tulpen</option>
   <option>Nelken</option>
</optgroup>
<optgroup label="B&auml;ume">
   <option>Buchen</option>
   <option>Eichen</option>
   <option>L&auml;rchen</option>
</optgroup>
</select>
```

Abbildung 14.76 Selectbox mit Optionsgruppen

14.72 small

Präsentationselement: Das Element `<small>` dient dazu, den eingeschlossenen Text auf ca. 80 % Schriftgröße zu verkleinern – er kann zur Erhöhung der Wirkung verschachtelt werden.

14.72.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext	Das Element `<small>` ist überall in Blockelementen erlaubt
Erlaubte Inhalte	Text und Inlinelemente, darunter auch `<small>` selbst
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01, XHTML 1.0, 1.1; **abgeschafft in:** XHTML 2.0
Mobil-Spezifikationen	XHTML mobile profile *nicht enthalten in:* HTML 4.0 mobile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<small> ... </small>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
small    { font-size: .83em }
```

14.72.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–

Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.72.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.72.4 Erläuterung zum Element

Das Element `<small>` bietet eine simple Möglichkeit, eingeschlossene Textpassagen gegenüber dem umliegenden Text leicht verkleinert darzustellen – seine Wirkung ist dabei durch Attribute nicht weiter steuerbar. Der Tag ist überall in Blockelementen einsetzbar.

Bei einer Verschachtelung mehrerer `<small>`-Container kaskadiert sich deren Wirkung – der Verkleinerungsfaktor des innen liegenden Containers bezieht sich in diesem Fall auf den bereits verkleinerten Inhalt des äußeren Containers.

Im folgenden Beispiel dient `<small>` der Präsentation eines Textes in Kapitälchen:

```
<p>Die Darstellung von K<small>APITÄLCHEN</small> ist alternativ
auch mit CSS möglich.</p>
```

Die Wirkung im Browser ist durchaus überzeugend:

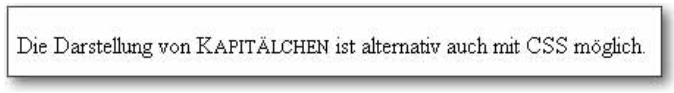

Abbildung 14.77 Kapitälchen mit <small>

Die Alternative mit CSS ist zwar relativ einfach, gelingt aber nur nach Anpassung des Quellcodes vollständig. Die Stylevorschrift bedient sich des `font-variant`-Properties:

```
span.sc { font-variant:small-caps;}
```

Der Versuch der unmittelbaren Anwendung auf den oben verwendeten Quelltext misslingt jedoch zunächst; alle Buchstaben des Wortes werden als normale Großbuchstaben dargestellt:

```
<p>Die Darstellung von <span class="sc">KAPITÄLCHEN</span> ist
alternativ auch mit CSS möglich.</p>
```

Abbildung 14.78 Fehlversuch von Kapitälchen mit CSS

Der Grund besteht darin, dass sich Kapitälchen per CSS nur aus Kleinbuchstaben generieren lassen. Daher muss das Rohmaterial für ein befriedigendes Ergebnis[48] entsprechend geändert werden:

```
<p>Die Darstellung von <span class="sc">Kapitälchen</span> ist
alternativ auch mit CSS möglich.</p>
```

Diesem Verhalten liegt durchaus eine Logik zugrunde – immerhin handelt es sich bei Kapitälchen um eine reine Präsentation, die eben auch beliebig anders erfolgen können soll (Textverarbeitungen handeln hier nicht anders; auch dort wird in Kapitälchen gesetzter Text nicht auf der Zeichenebene umgewandelt, sondern lediglich andere Darstellungsglyphen gewählt).

Analog zu dem bei `<big>` geschilderten Verhalten ist die untere Grenze einer mit `<small>` möglichen Verkleinerung abhängig vom Browser. Auch hier bindet der Internet Explorer das Verhalten stark an den `<font>`-Tag und die eingestellte Basisschriftgröße. So tritt dort bei der üblichen Basisschriftgröße von 3 ab doppelter Verschachtelung von `<small>`-Elementen keine weitere Verkleinerung mehr auf (die Implementierung entspricht wiederum dem `size`-Attribut von

48 Dies nützt bei älteren Versionen des Internet Explorers leider nichts – das Wort wird trotzdem komplett in Großbuchstaben dargestellt.

<font>; bei size="+2" ist die Untergrenze von Schriftgröße 1 erreicht). In Überschriften erster Ordnung (<h1>) mit größerer Schrift wäre eine entsprechend höhere Verschachtelungstiefe möglich.

Andere Browser (Firefox, Mozilla) nehmen die Verkleinerung auf em-Basis vor – die Verschachtelungstiefe ist hier zwar theoretisch beliebig, die Grenze der Lesbarkeit jedoch sehr schnell erreicht.

14.73 span

Das Element stellt (analog zum neutralen Blockcontainer) einen neutralen Inlinecontainer ohne festgelegte Präsentation dar. Er wird zusammen mit CSS-Regeln eingesetzt, um Textbereiche zu formatieren

14.73.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Überall in Block- und Inlineelementen, auch in selbst
Erlaubte Inhalte	Inlineelemente und Text (#PCDATA)
Offizielle W3C Spezifikationen	HTML 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<span> ... </span>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
span  {display:inline; }
```

14.73.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforeeditfocus, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfilterchange, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Databinding-, Edit-Attribute

14.73.3 Offizielle Attribute – W3C

class

Wert	*"classname [classname2 classname3 ... classnameN]"*
Typ	Ein, oder mehrere CSS-Klassenbezeichner
Verwendung	Optional

Das `class`-Attribut gehört zwar zu den allgemein verfügbaren Universalattributen, besitzt aber in Zusammenhang mit `<span>` eine besondere Bedeutung, da die wesentliche Aufgabe des Elements darin besteht, dem markierten Bereich eine CSS-Präsentation zuzuweisen.

Das Attribut `class` dient zum Binden einer Stylesheetklasse an den `<span>`-Container und erhält als Wert den Bezeichner einer solchen Klasse bzw. (falls mehrere Klassen gleichzeitig auf das Element angewendet werden sollen) eine durch Leerzeichen getrennte Liste mehrerer Klassenbezeichner.

id

Wert	*"ID"*
Typ	Ein eindeutiger Bezeichner vom Typ `ID`
Verwendung	Optional

Das id-Attribut kann zum Binden einer CSS-Präsentationsanweisung mittels eines ID-Selektors an ein -Element dienen. Dies kann von Interesse sein, wenn der -Container einen logischen Dokumentbereich markiert, der nur einmal im Dokument auftritt.

Anmerkung: Sofern es sich hierbei nicht um einen Inline-Container handeln muss, ist für den genannten Zweck jedoch der Einsatz eines -Containers gebräuchlicher.

style

Wert	*"CSS-Rule; [CSS-Rule2; CSS-Rule3; ... CSS-RuleN;]"*
Typ	Eine oder mehrere CSS-Anweisungen
Verwendung	Optional

Mittels des ebenfalls universell verfügbaren style-Attributs kann einem -Container direkt ein Inline-Style zugewiesen werden.

Anmerkung: Da der präsentationsneutrale -Container dadurch in die Rolle eines Präsentationselements gedrängt und die Trennung von Struktur und Präsentation unterlaufen wird, ist die alternative Zuweisung einer CSS-Klasse durch das class-Attribut vorzuziehen.

14.73.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datafld = "feldname"` (IE 4)

`dataformatas = "text" | "localized-text" | "HTML"`

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.73.5 Erläuterung zum Element

Das Element `<span>` dient als neutraler Inlinecontainer (*grouping element*) ohne festgelegte semantische Bedeutung. Es stellt somit ein Pendant zum ebenfalls neutralen Blockcontainer `<div>` dar.

Der einzige Zweck von `<span>` besteht darin, Textbereiche zu begrenzen und ihnen bestimmte Attribute zuzuweisen. Dies kann eine CSS-Präsentationsangabe sein oder wie hier eine Sprachangabe:

```
<p>Sie hat das gewisse <span lang="fr">je ne sais
 quoi</span>, findest du nicht?</p>
```

In Verbindung mit CSS-Klassen kann `<span>` auch Aufgaben übernehmen, für die man normalerweise XML einsetzen könnte, nämlich die Auszeichnung der *Bedeutung* eines Inhaltes:

```
<p>Der Kraftstofffilter <span class="produktname">BigX</span>
 hat die Bestellnummer <span class="bstnr">WF8200</span>.
 Er kostet <span class="preis_euro">50,70</span>EUR.</p>
```

Einerseits können so spezifische CSS-Präsentationen an die Inhalte gebunden werden, andererseits können die `class`-Attribute auch als Grundlage für Filterfunktionen dienen. (Ob diese Art des Einsatzes des `class`-Attributs nicht eigentlich eine Zweckentfremdung darstellt, sei hier dahingestellt.)

14.74 strike

Deprecated: Das Präsentationselement `<strike>` präsentiert den eingeschlossenen Inhalt durchgestrichen (*strikethrough*). Alias zu `<s>`.

14.74.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Inlinecontainer
XHTML-Modul	Legacy
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 3.2, 4.0/4.01 (tr/fs); XHTML 1.0 (tr/fs); *deprecated in:* HTML 4.0/4.01 strict, XHTML 1.1, 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikatione enthalten)

Syntaxbeispiel:

```
<strike> ... </strike>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
strike    { display: inline;
            text-decoration: line-through; }
```

14.74.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	–
Internet Explorer	`disabled`, *Access-, Edit-Attribute*

14.74.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.74.4 Erläuterung zum Element

Das Element `<strike>` präsentiert seinen Containerinhalt als durchgestrichen. Wie auch das Element `<s>` gilt es als »deprecated« und sollte nicht weiter verwendet werden. Nähere Erläuterungen finden Sie ab Seite 1 in der Beschreibung zu `<s>`.

Hinweis: Falls Sie eine Durchstreichung im Sinne einer *Bearbeitung* markieren wollen, stehen Ihnen hierfür die Elemente `<del>` und `<ins>` zur Verfügung, auf die Sie in diesem Falle zurückgreifen sollten. **[«]**

14.75 strong

Logische Auszeichnung: Der `<strong>`-Container markiert seinen Inhalt als »stark betont« (*strong*) im Sinne einer Hervorhebung.

14.75.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	HTML 4.0 mobile XHTML mobile profile *nicht enthalten in:* iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<strong> ... </strong>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
strong   { display:inline; font-weight:bold; }
```

14.75.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`accesskey`, `charset`, `coords`, `href`, `hreflang`, `methods`[49], `name`, `rel`, `rev`, `shape`, `tabindex`, `target`, `title`, `type`, `urn`[49]
Internet Explorer	Access-, Edit-Attribute

14.75.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

49 Abgeschafft.

14.75.4 Erläuterung zum Element

Das Element dient der Hervorhebung seines Containerinneren als »stark betont«. Gewöhnlich wird der Inhalt fett präsentiert (hiermit ist das Element ein logisches Pendant zu <b>).

```
<p>Das hier ist nur <b>fett</b>.
Das hier ist <strong>stark betont</strong>:</p>

<p>Ich m&ouml;chte <strong>so ungeheuer</strong> gern HTML
lernen!</p>

<p>Alle sagen, das sei <strong title="Wer&apos;s glaubt!">
total einfach</strong>!</p>
```

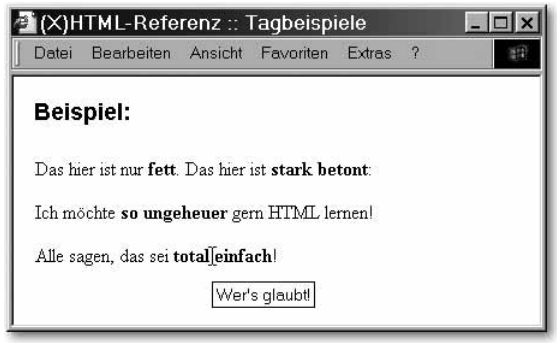

Abbildung 14.79 Fettauszeichnung und Hervorhebung im Vergleich; Tooltip mit title-Attribut

14.76 style

Der <style>-Container enthält Präsentationsregeln in einer Stylesheetsprache (derzeit ausschließlich CSS), die dazu dienen, das Dokument zu formatieren.

14.76.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Container, Headkontext
XHTML-Modul	Style Sheet
Kontext	Im Dokumentkopf <head>
Erlaubte Inhalte	#CDATA (Stylesheetanweisungen) in HTML #PCDATA in XHTML

Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<style type="text/css"> ... </style>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
style  { display: none; }
```

14.76.2 Attribute (Übersicht)

Standardattribute

HTML	`dir`, `lang`, `title`
XHTML	`dir`, `xml:lang`, `title`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	`onerror`, `onload`, `onreadystatechange`

Elementspezifische Attribute

W3C	`media`, **`type`**
Internet Explorer	`disabled`

14.76.3 Offizielle Attribute – W3C

media

Wert	*cdata*
Typ	ein Medientyp-Bezeichner
Verwendung	Optional

Das media-Attribut bezeichnet in Form einer kommagetrennten Liste den oder die Medientyp(en), die als Zielplattform der Präsentationsanweisungen dienen.

Hiermit kann die Anwendung der Styleanweisungen für ein bestimmtes Ausgabemedium gekennzeichnet werden. Gängige Werte sind screen, print, projection, handheld, aural und all.

- all
  Die Styleanweisungen werden für alle Medientypen angewendet, also grundsätzlich *gemeinsam* mit allen anderen anwendbaren Anweisungen.
- aural
  Barrierefreiheit: Die Styleanweisungen werden für Sprachausgabe angewendet.
- braille
  Barrierefreiheit: Die Styleanweisungen werden für Ausgabe mittels eines Braille-Lesegeräts angewendet.
- embossed
  Barrierefreiheit: Die Styleanweisungen werden zur Steuerung von Drukkern für Braille-Ausgabe eingesetzt.
- handheld
  Die Styleanweisungen werden für Ausgabe auf Mobiltelefonen, Organizern und vergleichbaren Ausgabegeräten eingesetzt.
- print
  Die Styleanweisungen werden für Druckausgabe eingesetzt.
- projection
  Die Styleanweisungen werden für Projektion über Beamer angewendet.
- **screen** (Default)
  Die Styleanweisungen werden für Bildschirmausgabe angewendet. Dieser Wert gilt defaultmäßig, wenn kein media-Attribut verwendet wird.
- tty
  Die Styleanweisungen werden bei Ausgabe mit textorientierten Browsern angewendet (eigentlich im Hinblick auf Fernschreiber eingeführt).
- tv
  Die Styleanweisungen werden für Ausgabe auf Fernsehgeräten angewendet, die sich technisch (Auflösung, Zeilenzahl etc.) von Computerbildschirmen unterscheiden.

Hinweis: Die Zuweisung an bestimmte Ausgabemedien kann innerhalb des <style>-Containers auch mittels der @import-Syntax vorgenommen werden: **[«]**

```
<style type="text/css">
   @import url(mobile.css) handheld;
</style>
```

type

Wert	*content-type*
Typ	Ein Content-Typ-Bezeichner
Verwendung	Obligatorisch

Das obligatorische `type`-Attribut nennt den Inhaltstyp des Stylecontainers, mit anderen Worten die Stylesprache. Es existiert kein Defaultwert. Das Attribut ist nicht *case-sensitive*. Für CSS ist der Wert `"text/css"` zu übergeben:

```
<style type="text/css">
   body {
        font-family: arial;
        }
</style>
```

14.76.4 Proprietäre Attribute – Internet Explorer

disabled

Wert	*"disabled"* | *"true"* | *"false"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Einführung	IE5

Das `disabled`-Attribut dient dazu, einen Stylecontainer im Internet Explorer zu deaktivieren und damit die Ausführung der in seinem Inneren getroffenen Präsentations- oder Importanweisungen zu unterbinden.

14.76.5 Erläuterung zum Element

Der `<style>`-Container dient als Behälter, um die zu HTML syntaxfremden CSS-Styleanweisungen als so genannten »Dokumentstyle« unmittelbar in ein zu präsentierendes HTML-Dokument einzubetten. Da der `<style>`-Container nur im Dokumentkopf erscheinen darf, wird dies auch als *header style information* bezeichnet.

Aufgrund des jeweils abweichenden Inhaltsmodells erfolgt die Einbettung von CSS innerhalb des `<style>`-Containers in HTML und XHTML unterschiedlich.

Inhaltsmodell von <style> in HTML

Das Inhaltsmodell von <style> in HTML ist CDATA:

```
<style type="text/css">
<!--    /* HTML-Kommentar verwendbar */
        p {font-family:arial}
-->
</style>
```

Das Auskommentieren des Elementinhalts richtet sich an alte, nicht stylefähige Browser. Diese würden die CSS-Anweisungen als Text darstellen, da sie den <style>-Container ignorieren. Ihre Sicht auf den CSS-Block wäre demnach folgende (der <style>-Container wird, da »unbekannt«, entfernt):

```
<!--    /* HTML-Kommentar verwendbar */
        p {font-family:arial}
-->
```

Die Vorgehensweise funktioniert, weil der stylefähige HTML-Parser im CDATA-Bereich keine Kommentaranweisungen beachtet, sondern den gesamten Inhalt bis zur abschließenden Endmarke </style> als Textdaten betrachtet[50].

Inhaltsmodell von <style> in XHTML

Das Inhaltsmodell von <style> in XHTML ist #PCDATA. Das SGML-Inhaltsmodell CDATA für Elemente ist in XML nicht vorgesehen[51], daher musste <style> in XHTML (ebenso wie <script>) mit dem Inhalt #PCDATA nachgebildet werden.

Es existieren eine Reihe von Möglichkeiten, in XHTML mit dem Inhalt des <style>-Containers umzugehen. Die alte Methode, diesen per HTML-Kommentar auszukommentieren, ist jedoch nicht mehr gangbar, sobald das Dokument vom Server als "application/xhtml+xml" ausgeliefert wird. In diesem Fall wird ein XML-konformer Parser innerhalb von #PCDATA-Bereichen auftretende Kommentare samt Inhalt entfernen und damit in diesem Fall die gesamten Styledefinitionen.

50 Ein orthodoxer SGML-Parser würde den Bereich beim Auftreten einer *beliebigen* Endmarke beenden, auch wenn diese nicht </style> lautet. Auch in der HTML-Spezifikation wird darauf hingewiesen, dass bereits die Zeichenfolge </ als Ende des Containers gewertet werden kann. Diese Zeichenfolge wird in CSS jedoch nicht auftreten.

51 XML ist nicht nur in dieser Hinsicht gegenüber SGML vereinfacht. Da HTML auf SGML aufbaut, XHTML jedoch auf XML, ergeben sich notwendigerweise Differenzen.

Kommentarverbot

In XHTML darf der Inhalt des `<style>`-Containers *nicht* durch einen HTML-Kommentar auskommentiert werden.

Die einfachste Vorgehensweise besteht darin, den Kommentar wegzulassen:

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

        p {font-family:arial}

</style>
```

Dies ist gefahrlos möglich, sofern die Styleanweisungen nicht das Zeichen < oder Zeichenfolgen enthalten, die als HTML-Steuerzeichen interpretiert werden können. Solche Zeichen müssten durch Ersetzen mit Entities (also `<` für <) ersetzt werden.

Ist dies zu aufwendig, kann der gesamte Containerinhalt in `CDATA`-Bereichsmarken eingeschlossen werden. Dies führt dazu, dass eventuelle Steuerzeichen lediglich als Zeichenliterale betrachtet werden:

```
<style type="text/css">
<![CDATA[

        p {font-family:arial}

]]>
</style>
```

Hier muss sichergestellt sein, dass im Inneren des Abschnitts nicht die Zeichenfolge `]]>` auftritt. Dieses Problem sollte jedoch selten bis nie auftreten. Im Zweifelsfall kann stets auf externe, verlinkte Stylesheets, die mit dem `<link>`-Element eingebunden werden, oder auf den Import von Stylesheets in den Stylecontainer per `@import`-Anweisung ausgewichen werden.

Liefert man das XHTML Dokument nicht als `"application/xhtml+xml"`, sondern als `"text/html"` aus, um ältere Browser zu bedienen, könnten diese allerdings mit den Marken `<![CDATA[` und `]]>` Probleme bekommen. Sie können jedoch gemäß der CSS-Kommentarsyntax auskommentiert werden:

```
<style type="text/css">
/* <![CDATA[ */

        p {font-family:arial}
```

```
/* ]]> */
</style>
```

Die entsprechenden Marken vor noch älteren Browsern zu verstecken und diesen gleichzeitig die HTML-Kommentare zuzuspielen, ist etwas aufwendig, aber machbar. Folgendes Konstrukt wäre hierfür erforderlich[52]:

```
<style type="text/css">
<!-- /* --><![CDATA[ /* > <!-- */

        p {font-family:arial}

/*]]>*/-->
</style>
```

Ein XHTML-Browser entfernt den ersten HTML-Kommentar und die darin enthaltene Startmarke des CSS-Kommentars, sieht also Folgendes:

```
<style type="text/css">
<![CDATA[ /* > <!-- */

        p {font-family:arial}

/* ]]> */-->
</style>
```

Ein alter HTML-Browser liest den Containerinhalt, überspringt (`CDATA`-Verhalten) den HTML-Kommentarbeginn, schaltet in den CSS-Modus, erkennt damit die Startmarke des ersten CSS-Kommentars und überspringt alles bis zum ersten CSS-Kommentarende. Die `CDATA`-Startmarke wird damit vor ihm versteckt, dasselbe erfolgt für die `CDATA`-Endmarke:

```
<style type="text/css">
<!-- /* CDATA-Start etc. */

        p {font-family:arial}

/* CDATA-Ende */  -->
</style>
```

Zum Schluss findet der Browser wie erwünscht das Ende des HTML-Kommentars.

52 Nach Ian Hickson, http://hixie.ch/advocacy/xhtml.

Externe Styledefinitionen

Um sich entsprechenden Ärger zu ersparen, greift man am Einfachsten auf externe CSS-Dateien zu. Dies geschieht jedoch nicht mittels des `<style>`-Containers (ein `src`-Attribut wie das bei `<script>` gibt es nicht), sondern mit dem `<link>`-Element:

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

Beachten Sie, dass hier ebenfalls der MIME-Typ genannt werden muss, sowie die Relation »Stylesheet« angegeben wird.

Kennzeichnung des MIME-Typs von Inlinestyles

Der MIME-Typ von Inlinestyles kann nicht unmittelbar bei der Styledefinition stattfinden, da diese ja im Starttag eines Elements über das `style`-Attribut erfolgt (es sei an dieser Stelle dahingestellt, ob Inlinestyles sinnvoll sind).

```
<!-- welche Stylesprache wird hier eingesetzt? -->
<p style="font-weight:bold">Das ist also fett.</p>
```

Die Lösung besteht darin, eine globale Default-Stylesprache zu definieren, die für alle Inline-Stylezuweisungen gilt. Dies geschieht mittels `<meta>`-Tag:

```
<meta http-equiv="Content-Style-Type" content="text/css"/>
```

[»] *Hinweis:* Ein analoges Vorgehen ist für Inlinescripte möglich, wie sie in Eventhandler-Attributen auftreten. Mehr über das `<meta>`-Element erfahren Sie ab Seite 1.

14.77 sub

Präsentationselement: Das Element `<sub>` (*subscript*) bewirkt die Tieferstellung seiner Inhalte und verkleinert sie um eine Schriftgröße (auf ca. 80 %).

14.77.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext	überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text

Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<sub> ... </sub>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
sub { font-size: .83em;
      vertical-align: sub; }
```

14.77.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.77.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.77.4 Erläuterung zum Element

Das Element dient der Tieferstellung seines Inhaltes innerhalb der Zeile.

```
<p>&quot;Subscript&quot;; Inhalt wird <sub>tiefgestellt</sub> und
eine Schriftgr&ouml;&szlig;e kleiner (auf ca. 80%) dargestellt.</p>

<p>H<sub>2</sub>O ist die chemische Formel für Wasser.</p>
```

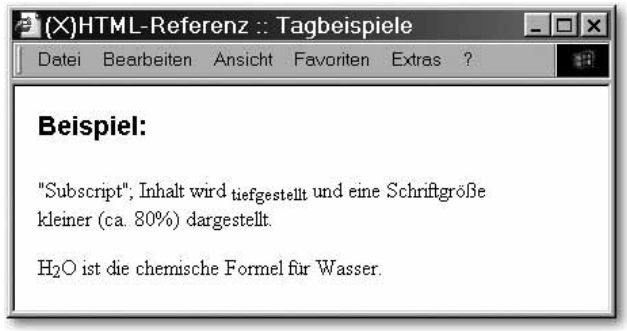

Abbildung 14.80 Tiefergestellter Text mit <sub>

14.78 sup

Präsentationselement: Das Element `<sup>` (*superscript*) bewirkt die Hochstellung seiner Inhalte und verkleinert sie um eine Schriftgröße (auf ca. 80 %).

14.78.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext	überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<sup> ... </sup>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
sup   { font-size: .83em; vertical-align: super }
```

14.78.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.78.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.78.4 Erläuterung zum Element

Das Element dient der Hochstellung seines Inhaltes innerhalb der Zeile.

```
<p>Inhalt wird <sup>hochgestellt</sup> und eine
Schriftgr&ouml;&szlig;e kleiner (auf ca. 80%) dargestellt.</p>

<p>r<sup>2</sup> ist die Kreisfläche.</p>

<p>Anm.: Optisch kann bei hochgestellter 2 ein Entity befriedigender
wirken:<br> r&sup2; (mit &amp;sup2;) ist die Kreisfl&auml;che.</p>
```

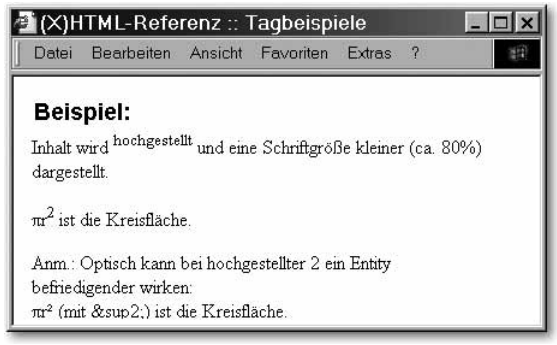

Abbildung 14.81 Hochstellung von Text mit <sup> im Vergleich mit Entity ²

14.79 table

Logische Auszeichnung: Der `<table>`-Container definiert einen Darstellungsblock in Form einer Tabelle (zeilenorientiert).

14.79.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Blockcontainer
XHTML-Modul	Basic Tables, Tables
Kontext	als Blockelement in `<body>`, `<div>` sowie in `<td>`, `<th>` zur Erzeugung verschachtelter Tabellenstrukturen
Erlaubte Inhalte	`<caption>`, `<thead>`, `<tbody>`, `<tfoot>`, `<tr>`
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic;
Mobil-Spezifikationen	XHTML mobile profile *nicht enthalten in:* HTML 4.0 mobile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<table> .... </table>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
table { display:table }
```

14.79.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfilterchange`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onscroll`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `bgcolor`, `border`, `cellpadding`, `cellspacing`, `datapagesize`, `frame`, `rules`, `summary`, `width`
Internet Explorer	`background`, `bordercolor`, `bordercolordark`, `bordercolorlight`, `clear`[53], `height`, `nowrap`[53], `valign`[53] *Access-, Databinding-, Edit-Attribute*
Netscape	`background`, `bordercolor`, `cols`, `height`, `hspace`, `vspace`

14.79.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `align`-Attribut erlaubt es, die Tabelle innerhalb des für sie definierten Blockbereiches zu zentrieren, oder aber als Float-Objekt im Zeilenkontext am entsprechenden Rand des übergeordneten Containers zu positionieren. Das Attribut wird vom W3C zugunsten von CSS-Anweisungen abgelehnt.

Die Defaultausrichtung der Tabelle ist linksbündig innerhalb des durch sie erzeugten Blocks. Eine analoge rechtsbündige Ausrichtung ist (ausgehend von der Tabelle) nicht möglich. Entsprechende Präsentationsanweisungen können jedoch in Zusammenhang mit dem übergeordneten Container getroffen werden.

- `align="center"`
  zentriert die Tabelle in ihrem Blockbereich.
- `align="left"`
  Float-Eigenschaft: Die Tabelle rückt an den linken Rand des umgebenden Blockelements, wird also, sofern er nicht die gesamte verfügbare Breite einnimmt, auf der rechten Seite von anderen Containern umflossen.
- `align="right"`
  Float-Eigenschaft: Die Tabelle rückt an den rechten Rand des umgebenden Blockelements, wird also, sofern er nicht die gesamte verfügbare Breite einnimmt, auf der linken Seite von anderen Containern umflossen

Anmerkung: Als historische Werte wurden im Internet Explorer auch `"bleedleft"`, `"bleedright"` und `"justify"` unterstützt. Hierbei setzte `"justify"` die

53 Abgeschafft seit IE 4 (nur IE 3).

Tabelle auf die gesamte, im Block verfügbare Breite, die Bleed-Werte bewirkten eine Links- und Rechtsausrichtung analog zu `"left"` und `"right"`, die sich jedoch am Dokumentrand und nicht am übergeordneten Container orientierte.

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `bgcolor` bestimmt die Hintergrundfarbe der Tabelle, wobei auch das Tabellengitternetz gefärbt wird, sofern dieses sichtbar ist. In Abwesenheit des Attributs gilt die Hintergrundfarbe des Dokuments.

Im *strict*-Vokabular ist das Attribut `bgcolor` nicht enthalten.

CSS-Alternative: `table {background-color:#rrggbb;}`

Beispiel: **[zB]**

```
<table width="300" bgcolor="#aaaaaa" border="1">
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
  ...
</table>
```

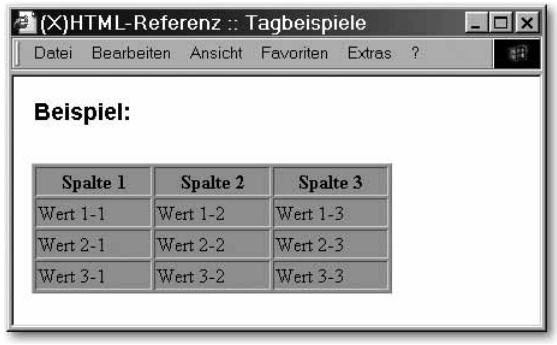

Abbildung 14.82 Tabelle mit Hintergrundfarbe (Darstellung mit CSS analog)

border

Wert	*number*
Typ	Ein numerischer Wert, als Pixelwert interpretiert
Verwendung	Optional

Das `border`-Attribut bestimmt die Breite in Pixel eines um die Tabelle gezeigten Rahmens. Der Defaultwert ist `"0"`, es wird also kein Rahmen gezeigt (und kein Tabellengitternetz).

[»] *Hinweis:* Ein Wert ungleich `"0"` bewirkt sowohl die Sichtbarkeit des Tabellenrahmens als auch des Tabellengitternetzes. Die Präsentation des Rahmens erfolgt für Werte gleich oder größer `"2"` mit 3-D-Effekt, also mit hellen und dunklen Randlinien. Als Default-Lichtfarbe wird in der Regel `#C6C3C6`, als Default-Schattenfarbe `#8C8E8C` verwendet.

Tipp

Soll gezielt die Sichtbarkeit von Teilen des Tabellenrahmens oder des Tabellengitternetzes festgelegt werden, so kann auf die Attribute `frame` (Tabellenrahmen) und `rules` (Tabellengitternetz) zurückgegriffen werden. Ist `border` auf den Wert `0` gesetzt, so entspricht die implizit `frame="void"` und `rules="none"`. Werden die Attribute `rules` und `frame` eingesetzt, so braucht `border` nur verwendet werden, wenn hierfür Werte ungleich `0` benötigt werden.

[zB] *Beispiel:*

```
<table width="300" border="8">
<caption>border="8"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

Abbildung 14.83 Tabellenrahmen von 8 Pixel Breite (Licht- und Schattenfarbe)

cellpadding

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `cellpadding`-Attribut legt für alle Tabellenzellen den Abstand zwischen Inhalt und innerem Zellrand in Pixeln oder Prozent fest.

Hinweis: Der Defaultwert des `cellpadding` ohne gesetztes Attribut ist ungleich Null. Er beträgt abhängig vom User Agent in der Regel 2 Pixel. **[«]**

CSS-Alternative[54]: `td, th {padding: 2px;}`

cellspacing

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `cellspacing`-Attribut legt den Abstand zwischen den Inhaltszellen (mit anderen Worten die Breite der Gitterlinien) in Pixeln oder Prozent fest.

Hinweis: Der Defaultwert des `cellspacing` ohne gesetztes Attribut ist ungleich Null. Er beträgt abhängig vom User Agent in der Regel 1 Pixel. **[«]**

CSS-Alternative: `table {border-spacing: 1px;}`

frame

Wert	*"void"* | *"above"* | *"below"* | *"hsides"* | *"lhs"* | *"rhs"* | *"vsides"* | *"box"* | *"border"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `frame`-Attribut legt die Sichtbarkeit bestimmter (Teil-)Bereiche des Tabellenrahmens fest.

- `frame="void"` (Default)
  Es wird kein Tabellenrahmen gezeigt (*void* = »leer«).
- `frame="above"`
  Nur die obere Seite des Tabellenrahmens wird gezeigt.
- `frame="below"`
  Nur die untere Seite des Tabellenrahmens wird gezeigt.
- `frame="hsides"`
  Die obere und untere Seite des Tabellenrahmens wird gezeigt (*horizontal sides*).

54 CSS-Padding wirkt zwar auch an der Tabelle, jedoch nicht wie gewünscht. Da das Property nicht vererbbar ist, muss es auf die Zellen selbst angewendet werden.

- `frame="vsides"`
  Die linke und rechte Seite des Tabellenrahmens wird gezeigt (*vertical sides*).
- `frame="lhs"`
  Die linke Seite des Tabellenrahmens wird gezeigt (*left hand side*).
- `frame="rhs"`
  Die rechte Seite des Tabellenrahmens wird gezeigt (*right hand side*).
- `frame="box"`, `frame="border"`
  Alle vier Seiten des Tabellenrahmens werden gezeigt.

[zB] *Beispiel:*

Eine Tabelle mit sichtbaren horizontalen Außenlinien (oben und unten):

```
<table width="300" frame="hsides">
<caption>frame="hsides"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

Eine Tabelle mit sichtbaren vertikalen Außenlinien (links und rechts):

```
<table width="300" frame="vsides">
<caption>frame="vsides"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

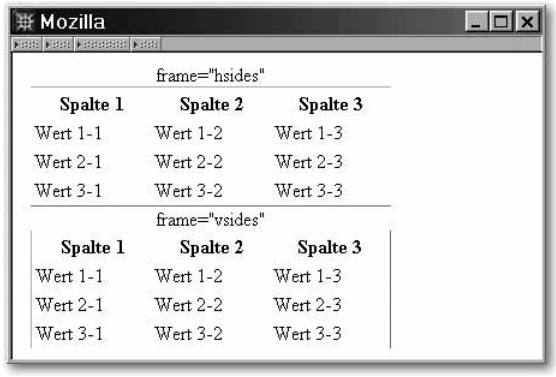

Abbildung 14.84 Tabellen mit horizontalen bzw. vertikalen Außenlinien

Der Wert `frame="box"` (alternativ kann auch `frame="border"` verwendet werden) blendet die Außenlinien auf allen vier Seiten ein:

```
<table width="300" frame="box">
<caption>frame="box"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

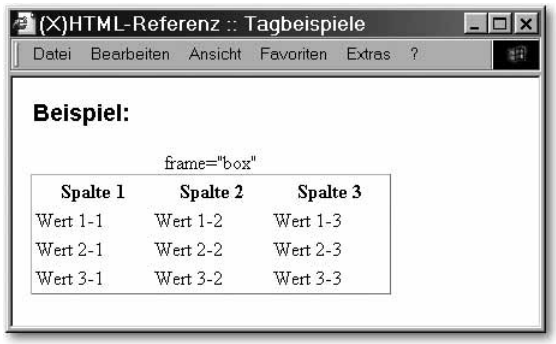

Abbildung 14.85 Tabelle mit allen vier Außenlinien

rules

Wert	*"none"* | *"groups"* | *"rows"* | *"cols"* | *"all"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `rules`-Attribut legt fest, welche (Teil-)Bereiche des Tabellengitternetzes zwischen den Zellen gezeigt werden sollen.

- `rules="none"` (Default)
  Das Tabellengitternetz ist vollständig unsichtbar.
- `rules="groups"`
  Nur das Gitternetz zwischen Zellengruppen, wie sie (horizontal) durch `<thead>`, `<tbody>` oder `<tfoot>` und (vertikal) durch `<colgroup>`-Container bestimmt werden, wird gezeigt.
- `rules="rows"`
  Das Tabellengitternetz zwischen den Zeilen ist sichtbar.
- `rules="cols"`
  Das Tabellengitternetz zwischen den Spalten ist sichtbar.

- `rules="all"`
  Das Tabellengitternetz ist in seiner Gesamtheit sichtbar.

[zB] *Beispiel:*

Diese Tabelle wird mit vollständigen innerem Gitternetz präsentiert:

```
<table width="300" frame="void" rules="all">
<caption>rules="all"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

Die folgende Tabelle erhält Linien zwischen den Tabellenzeilen:

```
<table width="300" frame="void" rules="rows">
<caption>rules="rows"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

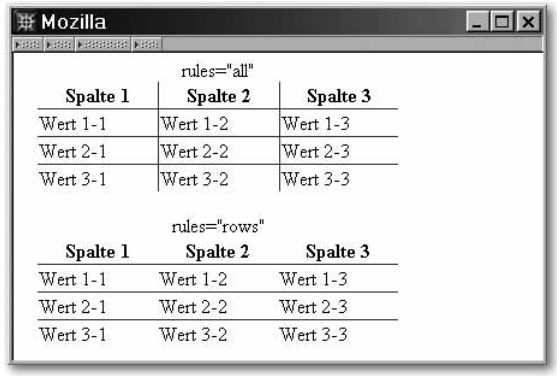

Abbildung 14.86 Tabellen mit rules="all" und rules="rows"

Diese Tabelle erhält durch das `rules`-Attribut Linien zwischen den Spalten, sowie durch das `frame`-Attribut an den Tabellenseiten:

```
<table width="300" frame="vsides" rules="cols">
<caption>frame="vsides"  rules="cols"</caption>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
```

```
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
<tr><td>Wert 3-1</td><td>Wert 3-2</td><td>Wert 3-3</td></tr>
</table>
```

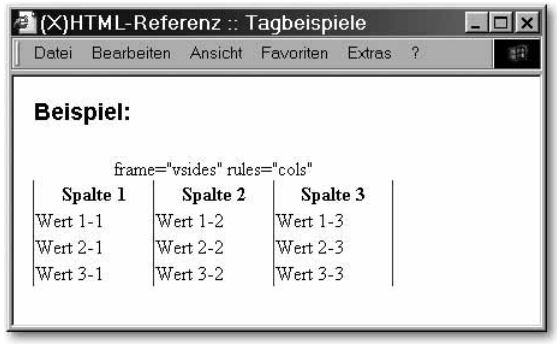

Abbildung 14.87 Tabelle mit Spalten- und Seitenlinien

summary

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Barrierefreiheit: Das `summary`-Attribut enthält einen Erklärungstext im Sinne einer inhaltlichen Zusammenfassung (*summary*). Dieser kann beispielsweise für User Agents mit Sprachausgabe oder ohne Tabellenfähigkeit eine alternative Beschreibung des Tabelleninhalts oder des Zwecks der Tabelle darstellen. Das Attribut ist *case-sensitive*.

Achtung: Das `summary`-Attribut wird im Internet Explorer nicht unterstützt. **[!]**

width

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `width`-Attribut legt die Breite der Tabelle als Pixelwert oder in Prozent des durch den übergeordneten Container zur Verfügung gestellten Wertes an. Eine Pixelangabe führt zu einer statischen Tabellenbreite. Eine Prozentangabe führt zu einer dynamischen Breite der Tabelle, wenn der übergeordnete Container ebenfalls eine dynamische Breite aufweist (beim `<body>`-Element ist das der Fall).

[»] *Hinweis:* Eine Breitenangabe ist nur dann relevant, wenn die so bestimmte Breite der Tabelle diejenige übersteigt, die durch die Breite der Zelleninhalte erzwungen wird. Die Tabelle kann auf keinen Fall schmaler dargestellt werden, als ihre Inhalte es erfordern.

14.79.4 Proprietäre Attribute – Internet Explorer

background

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `background` nennt für den Internet Explorer den URL einer Hintergrundgrafik für die Tabelle (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal hinter dem gesamten Tabellenbereich gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt eine einheitliche Rahmenfarbe des Tabellenrands. In Abwesenheit des Attributs gelten für den Tabellenrand die Default-Lichtfarbe (meist `#C6C3C6`) und die Default-Schattenfarbe (meist `#8C8E8C`).

bordercolordark

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolordark` bestimmt die dunkle untere Rahmenfarbe (Schattenfarbe) des Tabellenrands. In Abwesenheit des Attributs gilt die Default-Lichtfarbe (meist `#C6C3C6`) und die Default-Schattenfarbe (meist `#8C8E8C`) des Tabellenrands.

bordercolorlight

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolorlight` bestimmt die helle obere Rahmenfarbe (Lichtfarbe) des Tabellenrands. In Abwesenheit des Attributs gilt die Default-Lichtfarbe (meist `#C6C3C6`).

clear

Wert	*"left"* | *"right"* | *"all"* | *"no"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Gültigkeit	IE 3 (abgeschafft in IE 4)

(Historisch) Das `clear`-Attribut erzwingt die Darstellung der Tabelle nach einem Float-Objekt (beispielsweise eines Bildes oder einer Tabelle mit `align="left"` oder `align="right"` oder einem entsprechenden CSS-Property) zur nächsten freien Zeile des bezeichneten Rands des übergeordneten Blockelements hin. Der Vorschlag zu diesem Attribut stammt ursprünglich aus HTML 3.0. Es wurde nur kurzzeitig unterstützt.

CSS-Alternative: `table#table_id { clear:both; }`

height

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `height`-Attribut gibt die Höhe der Tabelle als Pixelwert oder in Prozent der Höhe des Browserfensters an.

Achtung: Das `height`-Attribut ist *nicht* Teil der offiziellen Spezifikation und wird daher nicht von allen Browsern zuverlässig interpretiert. **[!]**

nowrap

Wert	*"true"* | *"false"* | *"nowrap"*
Typ	Festgelegte Bezeichner
Verwendung	Optional
Gültigkeit	IE3 (abgeschafft in IE4)

(Historisch) Das `nowrap`-Attribut erlaubt bzw. unterdrückt den automatischen Zeilenumbruch innerhalb des Elements. Defaultmäßig bricht der Browser lange Zeilen um. Der Wert `nowrap="false"` gestattet ihm dies explizit, während `nowrap="true"` bzw. `nowrap="nowrap"` den Umbruch verbieten. Gegebenenfalls erscheint hierdurch eine horizontale Scrollleiste.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	festgelegte Bezeichner
Verwendung	Optional
Gültigkeit	IE3 (abgeschafft in IE4)

(Historisch) Das `valign`-Attribut bestimmt im Internet Explorer 3.0 die vertikale Ausrichtung der Inhalte aller Zellen der Tabelle. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zelle.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Databinding-Attribute

`datapagesize = "number"`

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.79.5 Proprietäre Attribute – Netscape

background

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `background` nennt den URL einer Hintergrundgrafik für die Tabelle (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich. Das Attribut entspricht in seiner Wirkung dem gleichnamigen Attribut beim Internet Explorer.

Anmerkung: Netscape 4.x ordnet eine Hintergrundgrafik fälschlicherweise nicht der Tabelle selbst insgesamt, sondern einzeln jeder Tabellenzelle zu.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt eine Rahmenfarbe des Tabellenrands. In Abwesenheit des Attributs gelten für den Tabellenrand die Default-Lichtfarbe (meist `#C6C3C6`) und die Default-Schattenfarbe (meist `#8C8E8C`).

cols

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Optional

(Historisch) Das `cols`-Attribut bietet eine Möglichkeit, dem Netscape-Browser zum Beschleunigen des Renderingvorgangs vor Einlesen des vollständigen Tabellenquellcodes mitzuteilen, wie viele Spalten die Tabelle aufweist.

Hinweis: Die beabsichtigte Wirkung ist standardkonform mittels der `<colgroup>`- und `<col>`-Elemente zu erzielen. **[«]**

height

Wert	*integerwert* | *prozentwert %*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Das `height`-Attribut gibt die Höhe der Tabelle als Pixelwert oder in Prozent der Höhe des Browserfensters an.

Achtung: Das `height`-Attribut ist nicht Teil der offiziellen Spezifikation und wird daher nicht von allen Browsern zuverlässig interpretiert. **[!]**

hspace

Wert	*integerwert*
Typ	Ganzzahl als Pixelwert
Verwendung	Optional
Gültigkeit	NN4

(Historisch) Das `hspace`-Attribut gibt die Breite eines symmetrischen horizontalen Freiraums links und rechts von der Tabelle an.

[!] *Achtung:* Das Attribut wird von neueren Netscape-Versionen sowie von Mozilla/Firefox nicht mehr unterstützt.

vspace

Wert	*integerwert*
Typ	Ganzzahl als Pixelwert
Verwendung	Optional
Gültigkeit	NN4

(Historisch) Das `vspace`-Attribut gibt die Breite eines symmetrischen vertikalen Freiraums über und unter der Tabelle an.

[!] *Achtung:* Das Attribut wird von neueren Netscape-Versionen sowie von Mozilla/Firefox nicht mehr unterstützt.

14.79.6 Erläuterung zum Element

Das `<table>`-Element definiert eine Tabelle, die laut W3C ausschließlich dazu gedacht ist, tabellarische Daten darzustellen. Die Endmarke `</table>` ist in HTML obligatorisch.

Der `<table>`-Container enthält als optionales erstes Element eine mit `<caption>` definierte Tabellenüberschrift. Auf die Überschrift dürfen Tabellenspaltendefinitionen mit `<colgroup>` oder `<col>` in beliebiger Anzahl folgen, an die sich als weitere optionale Elemente maximal je ein Tabellenkopfbereich `<thead>` und ein Tabellenfußbereich `<tfoot>` anschließt.

Der obligatorische Dateninhalt der Tabelle wird durch eine Folge von `<tbody>`-Containern gebildet, von denen (in HTML) mindestens einer vorhanden sein muss. Sind nur Tabellenzeilencontainer `<tr>` vorhanden, so darf der Browser das fehlende `<tbody>`-Element ergänzen.[55] In XHTML dagegen dürfen `<tr>` auch

55 Dies ist legal, da in HTML sowohl Start- als auch Endmarke von `<tbody>` entfallen dürfen.

unmittelbar in <table> erscheinen (jedoch nicht, wenn <tbody>-Container vorhanden sind).

Dies ist in XHTML daher eine legale Tabelle:

```
<table>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
</table>
```

In HTML ergänzt der Browser die obige Tabelle hingegen durch ein <tbody>-Element:

```
<table>
<tbody>
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
</tbody>
</table>
```

Barrierefreiheit

Sofern Sie noch Layouttabellen einsetzen (was nicht zeitgemäß ist), deklarieren Sie diese über das summary-Attribut entsprechend. Geben Sie Tabellenbreiten im Übrigen nicht statisch, sondern relativ zur Breite des Viewports an (beispielsweise durch Prozentangaben). Achten Sie darauf, dass die Tabelle linearisierbar ist, die Inhalte also eine sinnvolle Reihenfolge behalten, wenn sie durch einen Screenreader verarbeitet werden. (In der BITV, Abschnitt 5.1, sind die Ansprüche an barrierefreie Tabellen ausführlich beschrieben.)

14.80 tbody

Logisches Element: Das <tbody>-Element deklariert die Tabellenzeile(n), die der Container enthält, zum Tabellenkörper.

14.80.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Container
XHTML-Modul	Tables
Kontext	innerhalb des <table>-Containers

Erlaubte Inhalte	Tabellenzeilencontainer `<tr>`
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<tbody> ... </tbody>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
tbody      { display: table-row-group; }
```

14.80.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `char`, `charoff`, `valign`
Internet Explorer	`bgcolor`, *Access-, Edit-Attribute*

14.80.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte aller Zellen des Tabellenkörperbereichs. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet die Inhalte aller Zellen linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert die Inhalte aller Zellen innerhalb des Zellraumes.
- `align="right"`
  richtet die Inhalte aller Zellen rechtsbündig zum Zellenrand aus.
- `align="justify"`
  richtet die Inhalte aller Zellen blocksatzmäßig aus.
- `align="char"`
  richtet die Inhalte aller Zellen anhand eines Zeichens relativ zueinander aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann über das `char`-Attribut von `<tbody>` bestimmt werden.

  Achtung: Dieser Attributwert wird noch nirgends unterstützt. **[!]**

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem die Inhalte der Zellen des Tabellenkörperbereichs `<tbody>` durch `align="char"` gegeneinander ausgerichtet werden sollen. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Der Defaultwert ist je nach Spracheinstellung mittels `lang`-Attribut o.ä. der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Dieses Beispiel richtet die Inhalte der Zellen am Doppelpunkt aus:

```
<tbody align="char" char=":"> ... </tbody>
```

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent vom linken Zellrand der Zellen des Tabellenkörperbereichs für das Ausrichtungszeichen, das über das `char`-Attribut von `<tbody>` bezeichnet wird. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung der Inhalte aller Zellen des Tabellenkörperbereichs. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zelle.

- `valign="top"`
  richtet die Inhalte aller Zellen am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert die Inhalte aller Zellen vertikal im Zellraum.
- `valign="bottom"`
  richtet die Inhalte aller Zellen am unteren Zellenrand aus.
- `valign="baseline"`
  richtet die Inhalte nach einer je Zeile gemeinsamen Textbasislinie aus.

14.80.4 Proprietäre Attribute – Internet Explorer

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional
Einführung	IE4

Das Attribut `bgcolor` bestimmt die Hintergrundfarbe des Tabellenkörpers. In Abwesenheit des Attributs gilt die Hintergrundfarbe der Tabelle bzw. des Dokuments.

CSS-Alternative: `tbody {background-color:#rrggbb;}`

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.80.5 Erläuterung zum Element

Der `<tbody>`-Container gruppiert die Tabellenzeilen, in denen sich der eigentliche Tabelleninhalt befindet. Der Container ist in HTML innerhalb von Tabellen formal zwar obligatorisch, wird jedoch meist weggelassen, da sowohl Start- als auch Endmarke als auslassbar definiert sind. Nicht entfallen darf `<tbody>` jedoch, sobald ein Tabellenkopfbereich `<thead>` oder ein Tabellenfuß `<tfoot>` definiert werden, da deren Endmarken formal entfallen dürfen.[56]

In HTML ist die Endmarke `</tbody>` als *omittable* (auslassbar) definiert. In XHTML muss sie grundsätzlich gesetzt werden.

Pro Tabelle sind mehrere `<tbody>`-Gruppen erlaubt, wobei die erste Gruppe *nach* einer möglichen Tabellenüberschrift `<caption>`, *nach* eventuellen Spalten- oder Spaltengruppendefinitionen mit `<col>` oder `<colgroup>` und nach den optionalen Tabellenbereichen `<thead>` und `<tfoot>` stehen muss.

Als Inhalt von `<tbody>` sind ausschließlich Zeilencontainer `<tr>` zugelassen. Wird eine Zeilengruppierung mit `<tbody>` vorgenommen, so müssen sämtliche Tabellenzeilen, die nicht dem Tabellenkopf `<thead>` oder dem Tabellenfuß `<tfoot>` zugeordnet sind, Teil eines `<tbody>`-Bereiches sein.

56 In XHTML ist das Inhaltsmodell der Tabelle dahingehend geändert, dass alternativ eine Folge aus `<tbody>` oder `<tr>` zugelassen ist. In XHTML darf `<tbody>` also stets entfallen.

Die folgende Tabelle wäre in XHTML demnach nicht korrekt, weil die erste Zeile keiner Gruppe zugeordnet ist:

```
<table>
<!-- Zeile ohne Gruppe: -->
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tbody>
<tr><td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td></tr>
<tr><td>Wert 2-1</td><td>Wert 2-2</td><td>Wert 2-3</td></tr>
</tbody>
</table>
```

Die erste Tabellenzeile muss entweder ebenfalls mit einem `<tbody>`-Container umgeben werden oder (in diesem Falle sinnvoller) mit einem `<thead>`-Container (der HTML-Browser ergänzt für sich einen `<tbody>`).

Das `<tbody>`-Element besitzt von sich aus keine sichtbare Präsentation. Seine Begrenzung zur folgenden Tabellenzeilengruppe kann aber über das `rules`-Attribut des `<table>`-Containers sichtbar gemacht werden: `<table rules="groups">` erzeugt durchgezogene horizontale Linien zwischen allen Zeilengruppen.

Eine gesonderte Präsentation mittels CSS ist in gewissem Rahmen möglich, wobei lediglich Hintergrund- und Ausrichtungseigenschaften wirksam sind, da das CSS-Boxmodell auf `<tbody>` nicht anwendbar ist. Es ist jedoch möglich, mit `height` eine Höhe des Tabellenkörperbereiches anzugeben, wenn gleichzeitig das `overflow`-Property mit dem Wert `"auto"` oder `"scroll"` gesetzt wird – die Werte `"visible"` oder `"hidden"` sind nicht anwendbar (siehe Abbildung Abbildung 14.88).

[zB] *Beispiel:*

Ein scrollbarer Tabellenkörper, wie er derzeit jedoch nur in Mozilla erzeugt werden kann:

```
<table width="300" border="0" cellspacing="1">
  <thead style="background:#aaaaaa; color: white;">
    <tr>
        <th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th>
    </tr>
  </thead>
  <tfoot style="background:#aaaaaa; color: white;">
    <tr>
      <td>Fu&szlig;zeile 1</td>
      <td>Fu&szlig;zeile 2</td>
      <td>Fu&szlig;zeile 3</td>
    </tr>
```

```
  </tfoot>
  <tbody style="overflow: auto; height: 90px;">
    <tr>
      <td>Wert 11</td><td>Wert 12</td><td>Wert 13</td>
    </tr>
    <tr>
      <td>Wert 21</td><td>Wert 22</td><td>Wert 23</td>
    </tr>
    ...
  </tbody>
</table>
```

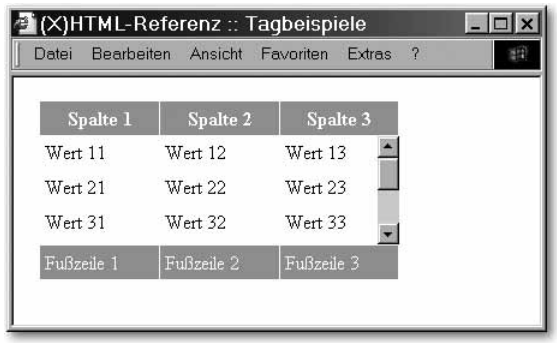

Abbildung 14.88 Mittels CSS-overflow wird der Tabellenkörper scrollbar.

14.81 td

Logisches Element: Definiert eine Tabellendatenzelle (*table data*) als Unterelement des Tabellenzeilencontainers `<tr>`

14.81.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Container
XHTML-Modul	Basic Tables, Tables
Kontext	In Tabellenzeilencontainer `<tr>`
Erlaubte Inhalte	Tabellen, Block-, Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	XHTML mobile profile *nicht enthalten in:* HTML 4.0 mobile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<td> ... </td>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
td          { display: table-cell; }
```

14.81.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	–
Internet Explorer Events	onafterupdate, onbeforecopy, onbeforecut, onbeforeeditfocus, onbeforepaste, onbeforeupdate, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfilterchange, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onrowenter, onrowexit, onselectstart

Elementspezifische Attribute

W3C	abbr, align, axis, bgcolor, char, charoff, colspan, headers, height, nowrap, rowspan, scope, valign, width
Internet Explorer	background, bordercolor, bordercolordark, bordercolorlight *Access-Attribute, Databinding-Attribute, Edit-Attribute*
Netscape	background

14.81.3 Offizielle Attribute – W3C

abbr

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Barrierefreiheit: Das `abbr`-Attribut dient dem Hinzufügen einer kurzen Beschreibung des Tabellenzelleninhalts oder dessen Bedeutung, die beispielsweise von Screenreadern bei der Ausgabe den Zelleninhalt vorangestellt werden kann. Das Attribut ist *case-sensitive*.

Anmerkung: Der Attributwert kann von visuell orientierten User Agents auch stellvertretend für den Zelleninhalt verwendet werden, wenn Platzgründe dies erfordern.

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt die horizontale Ausrichtung des Inhalts der Tabellenzelle. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet den Inhalt der Zelle linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert den Inhalt der Zelle innerhalb des Zellraumes.
- `align="right"`
  richtet den Inhalt der Zelle rechtsbündig zum Zellenrand aus.
- `align="justify"`
  richtet den Inhalt der Zelle blocksatzmäßig aus
- `align="char"`
  richtet den Inhalt der Zelle anhand eines Zeichens aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann über das `char`-Attribut von `<td>` bestimmt werden.

  Achtung: Dieser Attributwert wird noch nirgends unterstützt. **[!]**

axis

Wert	*"kategorie_a [, kategorie_b, ..., kategorie_n]"*
Typ	Eine kommagetrennte Liste aus Kategoriebezeichnern
Verwendung	Optional

Barrierefreiheit: Das `axis`-Attribut ordnet die Tabellenzelle einer oder mehreren Inhaltskategorien zu, um dem User Agent zu ermöglichen, Inhalte zu klassifizieren oder (im Rahmen einer Query) für bestimmte Ausgaben zu filtern. Eine typische Verwendung wäre der Einsatz der zugeordneten Kategorie(n) in einem Screenreader als Verschlagwortung des Zelleninhalts (vergleichbar mit dem `abbr`-Attribut).

[zB] *Beispiel:*

```
<table>
<tr>
<td axis="Buch, Literatur">Thomas Mann, &quot;Tod in
Venedig&quot;</td>
</tr>
<tr>
<td axis="CD, Literatur">Thomas Mann, &quot;Tod in Venedig&quot;
(H&ouml;rbuch)</td>
</tr>
<tr>
<td axis="CD, Musik">John Coltrane, &quot;A Love Supreme&quot;</td>
</tr>
<tr>
<td axis="Buch, Musik">Kahn/Hein, &quot;A Love Supreme&quot;</td>
</tr>
</table>
```

Anmerkung: Das `axis`-Attribut (seinerzeit jedoch im Plural mit `axes` bezeichnet) wurde bereits im Vorschlag zu HTML 3.0 konzipiert, um das Einlesen von HTML-Tabellen in Datenbanken zu erleichtern. Hierbei sollten die Bezeichner von Datenbankfeldern als Attributwerte verwendet werden. Gleichzeitig existierte auch ein (nicht identisches) Attribut `axis`, das von seiner Intention dem heutigen `abbr`-Attribut entsprach. Beide Attribute standen von Anfang an auch in Zusammenhang mit Sprachausgabe.

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `bgcolor` bestimmt die Hintergrundfarbe der Tabellenzelle. In Abwesenheit des Attributs gilt die Hintergrundfarbe der umgebenden Tabellenzeile `<tr>`, der Tabelle oder des Dokuments.

Im *strict*-Vokabular ist das Attribut `bgcolor` nicht enthalten.

CSS-Alternative: `td {background-color:#rrggbb;}`

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem der Inhalt der Tabellenzelle durch `align="char"` ausgerichtet werden soll. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Der Defaultwert ist je nach Spracheinstellung über das `lang`-Attribut oder ähnlich der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Dieses Beispiel richtet die Inhalte der Zelle am Doppelpunkt aus:

```
<td align="char" char=":"> ... </td>
```

Nicht unterstützt in Opera.

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent vom linken Zellrand für das über das `char`-Attribut des Elements bezeichneten Ausrichtungszeichen. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

Nicht unterstützt in Opera.

colspan

Wert	*number*
Typ	Positive Ganzzahl größer 1
Verwendung	Optional

Das colspan-Attribut (*column span*) dehnt den Inhaltsbereich der Tabellenzelle nach rechts über mehrere Tabellenspalten aus. Die Zahl der überspannten Spalten wird durch eine positive Ganzzahl angegeben, wobei die Ursprungsspalte mitgezählt wird. Die colspan-Zelle steht daher in ihrer Zeile für eine entsprechende Zahl von Zellendefinitionen.

[»] *Hinweis:* Übersteigt die Zahl der Zellendefinitionen durch ein colspan die Anzahl der real existierenden Zellen in anderen Zeilen, so wird dies dort durch Anfügen von Blindzellen am rechten Tabellenrand aufgefüllt.

```
<table>
<tr>
<!-- drei Zellendefinitionen nach rechts durch colspan: -->
<td colspan="3">B&uuml;cher von Thomas Mann</td>
</tr>
<tr>
<!-- drei reale Zellendefinitionen: -->
<td>Tod in Venedig</td>
<td>Buddenbrooks</td>
<td>Zauberberg, Der</td>
</tr>
</table>
```

Anmerkung: Eine Zelle kann sowohl colspan- als auch rowspan-Attribut besitzen und sich daher gleichzeitig über mehrere Spalten wie Zeilen erstrecken.

headers

Wert	*idrefs*
Typ	Leerzeichengetrennte Liste aus IDREFS
Verwendung	Optional

Barrierefreiheit: Das headers-Attribut verknüpft die Tabellenzelle logisch über IDREF-Beziehungen mit einer oder mehreren Tabellenheaderzellen, die den entsprechenden, als Teil einer IDREF-Liste übergebenen ID-Bezeichner besitzen. Die Beziehung kann im Rahmen der Sprachausgabe der Tabelle unterstützend eingesetzt werden. Das Attribut ist *case-sensitive*.

[zB] *Beispiel:*

```
<table>
<tr>
<th id="rose">Rosengew&auml;chse</th>
<th id="lilie">Liliengew&auml;chse</th>
<th id="nelke">Nelkengew&auml;chse</th>
```

```
</tr>
<tr>
<td headers="rose">Buschrose</td>
<td headers="lilie">Tulpe</td>
<td headers="nelke">Pfingst-Nelke</td>
</tr>
<tr>
<td headers="rose">Heckenrose</td>
<td headers="lilie">Tiger-Lilie</td>
<td headers="nelke">Gartennelke</td>
</tr>
</table>
```

Anmerkung: Eine Beziehung zwischen Daten- und Headerzelle wird ebenfalls durch das `scope`-Attribut hergestellt – dies jedoch in umgekehrter Richtung.

height

Wert	*integerwert* | *prozentwert*%
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `height`-Attribut bestimmt die Mindesthöhe der Tabellenzelle als Pixelwert (Ganzzahl) oder mit einer Prozentangabe. In Abwesenheit des Attributs wird die Höhe durch das `height`-Attribut der Tabellenzeile `<tr>` oder die Inhalte der Zelle bestimmt.

Anmerkung: Die Höhenangabe ist nur relevant, solange die Zelleninhalte und diejenigen der Nachbarzellen der gleichen Zeile dies zulassen. Die Zelle kann je nach den Abmessungen der enthaltenden Elemente (beispielsweise Grafiken) auch auf eine die Angaben überschreitende Höhe aufgespannt werden.

nowrap

Wert	*"nowrap"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `nowrap`-Attribut erlaubt bzw. unterdrückt den automatischen Zeilenumbruch innerhalb der Tabellenzelle. Defaultmäßig bricht der Browser lange Zeilen automatisch um. Der Wert `nowrap="nowrap"` verbietet dies. Gegebenenfalls erscheint durch die erhöhte Tabellenbreite eine horizontale Scrollleiste.

Anmerkung: Der Internet Explorer interpretiert zusätzlich die Werte `"true"` und `"false"`, wobei `nowrap="false"` den Umbruch explizit zulässt, während `nowrap="true"` ihn verbietet. Andere Browser unterstützen diese Werte nicht.

rowspan

Wert	*number*
Typ	Positive Ganzzahl größer 1
Verwendung	Optional

Das `rowspan`-Attribut dehnt den Inhaltsbereich der Tabellenzelle nach unten über mehrere Tabellenzeilen aus. Die Zahl der überspannten Zeilen wird durch eine positive Ganzzahl angegeben, wobei die Ursprungszeile mitgezählt wird. Die `rowspan`-Zelle steht daher in ihrer Spalte für eine entsprechende Zahl von Zellendefinitionen. Die realen Zellen der Folgezeile weichen den `rowspan`-Definitionen nach rechts aus.

[»] *Hinweis:* Übersteigt die Zahl der Zellendefinitionen durch ein `rowspan` die Anzahl der real existierenden Zellen in anderen Zeilen, so wird dies dort durch Anfügen von Blindzellen am rechten Tabellenrand aufgefüllt.

[zB] *Beispiel:*

```
<table>
<tr>
<!-- drei Zellendefinitionen nach unten durch rowspan: -->
<td rowspan="3">B&uuml;cher von Thomas Mann</td>
<td>Tod in Venedig</td>
</tr>
<tr>
<!-- hier gilt der rowspan aus Zeile 1-->
<td>Zauberberg, Der</td>
</tr>
<tr>
<!-- hier gilt der rowspan aus Zeile 1-->
<td>Zauberberg, Der</td>
</tr>
<tr>
<!-- ab hier rowspan beendet-->
<td> ... </td>
<td> ... </td>
</tr>
</table>
```

Anmerkung: Eine Zelle kann sowohl `colspan`- als auch `rowspan`-Attribut besitzen und sich daher gleichzeitig über mehrere Spalten wie Zeilen erstrecken.

scope

Wert	*"row"* | *"col"* | *"rowgroup"* | *"colgroup"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Barrierefreiheit: Das `scope`-Attribut nennt den Bezug der Tabellenzelle zu einem Bereich von Tabellendatenzellen. Im einfachsten Fall ist dieser Bezug spalten- oder zeilenorientiert und erstreckt sich auf die betreffende Spalte bzw. Zeile, in der sich die Tabellenzelle befindet. Ein Screenreader setzt beim Verlesen entsprechender Tabelleninhalte jeweils den Inhalt der zugeordneten Zelle voran.

- `scope="col"` (Default)
  Die Tabellenzelle nimmt Bezug auf ihre Tabellenspalte.
- `scope="row"`
  Die Tabellenzelle nimmt Bezug auf ihre Tabellenzeile.
- `scope="colgroup"`
  Die Tabellenzelle nimmt Bezug auf eine, mit `<colgroup>` definierte Spaltengruppe.
- `scope="rowgroup"`
  Die Tabellenzelle nimmt Bezug auf eine Zeilengruppe, wie sie mit `<thead>`, `<tbody>` oder `<tfoot>` definiert werden kann.

Anmerkung: Gewöhnlich wird das `scope`-Attribut eher ausgehend von Tabellenheaderzellen `<th>` eingesetzt.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung des Inhalts der Tabellenzelle. Defaultwert ist `"middle"`, also die vertikale Zentrierung des Inhalts innerhalb der Zelle.

- `valign="top"`
  richtet den Inhalt der Zelle am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert den Inhalt der Zelle vertikal im Zellraum.

- `valign="bottom"`
  richtet den Inhalt der Zelle am unteren Zellenrand aus.
- `valign="baseline"`
  richtet die Inhalte aller Zellen einer Zeile nach einer gemeinsamen Textbasislinie aus, sobald dort der Attributwert für eine Zelle gesetzt ist.

width

Wert	*integerwert* | *prozentwert%*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `width`-Attribut bestimmt die Mindestbreite der Tabellenzelle als Pixelwert (Ganzzahl) oder mit einer Prozentangabe. In Abwesenheit des Attributs wird die Breite durch die Breite der Inhalte der Zelle selbst oder der breitesten Zelle ihrer Spalte bestimmt.

Die Breitenabgabe ist nur relevant, solange die Zelleninhalte dies zulassen: Die Zelle kann je nach den Abmessungen der sie enthaltenden Elemente (beispielsweise Grafiken) auch auf eine, die Angaben überschreitende Breite aufgespannt werden.

14.81.4 Proprietäre Attribute – Internet Explorer

Internet Explorer

background

Wert	*URL*
Typ	Der URL einer Grafik, die als Hintergrund dienen soll
Verwendung	Optional

Das Attribut `background` nennt den URL einer Hintergrundgrafik für die Tabellenzelle (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich.

Übliche Grafikformate sind JPG und GIF, wobei auch transparente und (browserabhängig) animierte GIFs gestattet sind. Eine gleichzeitig mit `bgcolor` definierte Hintergrundfarbe gilt als *hinter* der Hintergrundgrafik befindlich – sie ist durch eventuelle transparente Bereiche einer Hintergrundgrafik sichtbar.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt die einheitliche Rahmenfarbe der Tabellenzelle. In Abwesenheit des Attributs gilt die Rahmenfarbe der umgebenden Tabellenzeile `<tr>`, diejenige der Tabelle oder die Defaultfarbe des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

bordercolordark

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolordark` bestimmt die dunkle obere Rahmenfarbe (Schattenfarbe) der Tabellenzelle. In Abwesenheit des Attributs gilt der Wert des `bordercolor`-Attributs, die Rahmenfarbe der umgebenden Tabellenzeile `<tr>`, diejenige der Tabelle oder die Default-Schattenfarbe (meist `#8C8E8C`) des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

bordercolorlight

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolorlight` bestimmt die helle untere Rahmenfarbe (Lichtfarbe) der Tabellenzelle. In Abwesenheit des Attributs gilt der Wert des `bordercolor`-Attributs, die Rahmenfarbe der umgebenden Tabellenzeile `<tr>`, diejenige der Tabelle oder die Default-Lichtfarbe (meist `#C6C3C6`) des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) | `"false"` | `"true"`

`unselectable = "on" | "off"` (IE 5)

14.81.5 Proprietäre Attribute – Netscape

background

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `background` nennt den URL einer Hintergrundgrafik für die Tabellenzelle (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich.

14.81.6 Erläuterung zum Element

Der `<td>`-Container bezeichnet eine gewöhnliche Tabellendatenzelle. Die Endmarke `</td>` ist in HTML als *omittable* (auslassbar) definiert. In XHTML muss die Endmarke grundsätzlich gesetzt werden. Das Element tritt ausschließlich als Kindelement des Tabellenzeilen-Containers `<tr>` auf.

Als Inhalte von `<td>` sind Text sowie Block- und Inlineelemente gestattet. Als alternative Beschreibung des Zelleninhalts kann das `abbr`-Attribut eingesetzt werden, das durch Screenreader oder visuelle User Agents mit verringertem Raum zur Darstellung der Tabelle anstatt des eigentlichen Inhalts der Zelle ausgewertet werden kann.

Eine Beziehung zwischen Datenzellen und den ihnen als Überschrift zuzuordnenden Headerzellen sind über eine `ID-IDREFS`-Beziehung herstellbar, wobei die Listenwerte des `header`-Attributs der Datenzelle auf die `id`-Attribute einer oder mehrerer Headerzellen `<th>` verweisen können.

Hinweise zur Barrierefreiheit

Verwenden Sie für Datentabellen die entsprechenden Attribute `headers` und `axis`, um die Beziehung zwischen Kopf- und Datenzellen zu verdeutlichen oder um weitere inhaltliche Informationen zu bieten. Stellen Sie für umfangreiche Zel-

leninhalte mit Hilfe des abbr-Attributs eine Abkürzung bereit. Setzen Sie auch das summary-Attribut für <table> ein, um eine Beschreibung der Tabelle zur Verfügung zu stellen. (In der BITV, Abschnitt 5.1, sind die Ansprüche an barrierefreie Tabellen ausführlich beschrieben.)

14.82 textarea

Formularelement. Der <textarea>-Container erzeugt ein mehrzeiliges Texteingabefeld, dessen Abmessungen über seine Attribute festgelegt werden.

14.82.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Basic Forms, Forms
Kontext	Überall in Block- und Inlineelementen innerhalb eines <form>-Containers
Erlaubte Inhalte	Text
Offizielle W3C Spezifikationen	HTML 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1; XHTML Basic **abgeschafft in:** XHTML 2.0
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<textarea> ... </textarea>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
textarea      { display:inline; }
```

14.82.2 Attribute (Übersicht)

Standardattribute

HTML	class, dir, id, lang, style, title
XHTML	class, dir, id, xml:lang, style, title

Eventhandler-Attribute

Common Events Collection	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Intrinsic Events Collection	onblur, onchange, onfocus, onselect
Internet Explorer Events	onafterupdate, onbeforecopy, onbeforecut, onbeforeeditfocus, onbeforepaste, onbeforeupdate, oncontextmenu, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onerrorupdate, onfilterchange, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onrowenter, onrowexit, onscroll, onselectstart

Elementspezifische Attribute

W3C	accesskey, **cols**, disabled, name, readonly, **rows**, tabindex
Internet Explorer	wrap *Access-Attribute, Databinding-Attribute, Edit-Attribute*
Netscape	wrap
DoCoMo iMode	istyle

14.82.3 Offizielle Attribute – W3C

accesskey

Wert	*[a-z][1–0]*#*
Typ	Alphanumerisches Zeichen, sowie * und #; nicht *case-sensitive*.
Verwendung	Optional

Barrierefreiheit: Das Attribut accesskey bestimmt im Sinne der Accessibility ein einzelnes alphanumerisches Tastaturzeichen, das im Sinne eines Keyboard-Shortcuts das Eingabefeld in den Fokus nimmt – unter Windows bei gleichzeitig gedrückter Alt-Taste[57], unter Mac OS mit Ctrl-Taste.

```
<textarea name="text" accesskey="t"> ... </textarea>
```

Das Textbereichsfeld im oberen Beispiel wird mit Alt+T ausgewählt.

57 Unter Windows in Opera-Browsern statt mit Alt mit gleichzeitig gedrückter ⇧- und Esc-Taste. Hintergedanke ist, den Tastaturzugang zum Browsermenü per Alt-Taste nicht zu behindern. Dieser bleibt jedoch bei einzelner Betätigung der Alt-Taste auch bei den anderen Browsern unbeschränkt möglich.

Hinweis: Auf Webseiten für mobile Endgeräte sollten als Accesskeys lediglich die Ziffern 0–9 verwendet werden. Des Weiteren sind theoretisch auch * (Asterisk) und # (Hash-Symbol) gestattet, wobei (geräteabhängig) die durch die entsprechenden Tasten gesteuerte Seitenscrollfunktion Vorrang vor den Accesskeys besitzt. **[«]**

cols

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Obligatorisch

Das `cols`-Attribut legt die Zahl der pro Zeile sichtbaren Zeichen des eingegebenen Textes fest und bestimmt damit indirekt die Breite des Texteingabefeldes. Übersteigt die Zeichenzahl der Texteingabe die angegebene Zeichenzahl, so kann ein horizontaler Scrollbalken eingeblendet werden.

Hinweis: Die Spezifikation erlaubt dem User Agent ausdrücklich, gegebenenfalls selbstständig an geeigneten Textstellen einen Zeilenumbruch vorzunehmen. **[«]**

disabled

Wert	*"disabled"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Das `disabled`-Attribut deaktiviert ein Textfeld und unterdrückt die Versendung seines Wertes beim Abschicken der Formulardaten. Gleichzeitig wird durch das Attribut verhindert, dass der User den Zustand des Elements ändert (durch Werteingabe oder Anklicken). Ein deaktiviertes Textfeld wird durch Ausgrauung präsentiert.

Beispiel: **[zB]**

```
<textarea disabled="disabled">Dieser Defaulttext kann nicht
überschrieben werden.</textarea>
```

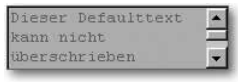

Abbildung 14.89 Ein deaktiviertes Textfeld mit Defaulttext

name

Wert	*nmtoken*
Typ	Ein Bezeichner in Form eines NMTOKEN
Verwendung	Optional

Das `name`-Attribut legt den Bezeichner (*control name*) des Textfeldes innerhalb eines Formulars fest. Der Bezeichner findet bei der Datenübertragung als name-Part eines »name-value«-Pärchens Verwendung. Der Attributwert ist nicht *case-sensitive*, daher werden beispielsweise `name="FELDNAME"` und `name="feldname"` als identisch betrachtet.

Der Gültigkeitsbereich von *control names* beschränkt sich auf das beherbergende Formular – in einem Dokument dürfen *control names* (im Gegensatz zu IDs) daher mehrfach erscheinen, solange sie zu verschiedenen Formularen gehören.

[»] *Hinweis:* Der Wert eines Textfeldes ohne `name`-Attribut wird beim Datenversand nicht berücksichtigt.

readonly

Wert	*"readonly"*
Typ	Festgelegter Bezeichner
Verwendung	Optional

Das `readonly`-Attribut kann eingesetzt werden, um eine Änderung eines Vorgabewertes durch den Nutzer zu unterbinden. Die Anwendung erstreckt sich im Wesentlichen auf Texteingabefelder, die scriptgesteuert Werte ausgeben, die der Nutzer zur Kontrolle betrachten können, aber nicht ändern soll.

[»] *Hinweis:* Die Werte von als `"readonly"` gekennzeichneten Elementen werden anders als im Falle von `"disabled"` in jedem Fall übertragen.

Anmerkung: Der Internet Explorer unterstützt für dieses Attribut auch die Werte `"true"` (nur Lesen) und `"false"` (kein Schreibschutz; Default).

rows

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Obligatorisch

Das `rows`-Attribut legt die Zahl der im Textfeld sichtbaren Zeilen des eingegebenen Textes fest und bestimmt damit indirekt die Höhe des Texteingabefeldes.

Übersteigt die eingegebene Textmenge die entsprechende Zeilenzahl, so wird ein vertikaler Scrollbalken gezeigt.

tabindex

Wert	*integerwert*
Typ	Positive Ganzzahl
Verwendung	Optional

Barrierefreiheit: Die an das Attribut `tabindex` übergebene Ganzzahl legt in aufsteigender Richtung die Position des Texteingabefeldes in der Tabulatorwahl-Reihenfolge (⇆-Taste) fest.

Die Tab-Reihenfolge gestaltet sich wie folgt:

- Zuerst die Elemente mit Tabindex, vom kleinsten verwendeten (positiven) Index in aufsteigender Reihenfolge. Elemente mit gleichem Index werden in Quelltextreihenfolge angesprochen.
- Aktivierbare Elemente ohne gesetztes `tabindex`-Attribut in Quelltextreihenfolge. (Zu den aktivierbaren Elementen gehören allgemein neben `<textarea>` auch `<a>`, `<area>`, `<button>`, `<input>`, `<object>` und `<select>`.)

Hinweis: Die Tab-Reihenfolge enthält alle aktivierbaren Elemente gemeinsam, also auch Links. Für Links und Formularelemente sollten daher niemals gleiche Tabindex-Werte gesetzt werden. **[«]**

14.82.4 Proprietäre Attribute – Internet Explorer

wrap

Wert	*"soft" | "hard" | "off"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

(Internet Explorer ab 4.0) Das `wrap`-Attribut bestimmt das Verhalten des Elements in Bezug auf den Umbruch des enthaltenen Textes.

- `wrap = "soft"` (Defaultwert)
  Der Text wird mit Umbruch dargestellt. Die Zeilenumbruchbefehle (CR und LF) werden jedoch nicht übertragen.
- `wrap = "hard"`
  Der Text wird mit Umbruch dargestellt. Die Zeilenumbruchbefehle (CR und LF) werden zusammen mit den Textdaten übertragen.

- `wrap = "off`
  Der Text wird ohne Umbruch dargestellt, wenn der Nutzer nicht selbst Zeilenumbruchbefehle vornimmt. In diesem Falle werden Zeilenumbruchbefehle übertragen.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute
`hidefocus = "true" | "false"` (ab IE 5.5)

IE-Databinding-Attribute
`datafld = "feldname"` (IE 4)

`datasrc = "datenquellen-id"` (IE 4)

IE-Edit-Attribute
`atomicselection = "true" | "false"` (IE 5)

`unselectable = "on" | "off"` (IE 5)

14.82.5 Proprietäre Attribute – Netscape

wrap

Wert	*"off"* | *"hard"* | *"soft"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

(Netscape Navigator ab 4.0) Das `wrap`-Attribut bestimmt das Verhalten des Elements in Bezug auf den Umbruch des enthaltenen Textes. Die Wirkung entspricht der des gleichnamigen Attributs des Internet Explorers.

14.82.6 Proprietäre Attribute – DoCoMo I-Mode

istyle

Wert	*"1"* | *"2"* | *"3"* | *"4"*
Typ	number
Verwendung	Optional
Einführung	iMode HTML 2.0

Mit dem `istyle`-Attribut wird der Eingabestil (alphabetisch, numerisch) eines `<textarea>`-Eingabebereiches in i-Mode kompatiblen Browsern festgelegt. Als Attributwerte sind die Zahlen von 1 bis 4 zugelassen. Von den Vorgaben abweichende Eingabewerte bewirken den Defaultstyle »Full size kana«.

- `istyle="1"` (Default)
  Full-size kana
- `istyle="2"`
  Half-size kana
- `istyle="3"`
  Alphabetic
- `istyle="4"`
  Numeric

14.82.7 Erläuterung zum Element

Innerhalb eines Formulars können Eingabefelder vom Typ `<textarea>` dort eingesetzt werden, wo die übersichtliche Eingabe größerer Textmengen über mehrere Zeilen erforderlich ist, beispielsweise für Kommentare oder Anfragen.

Der Elementinhalt von `<textarea>` ist als `#PCDATA` definiert. Im Containerinneren werden daher zwar Entities ersetzt, aber keine HTML-Tags interpretiert. Leerraum und Zeilenumbrüche innerhalb des Defaulttextes werden beachtet.

```
<textarea cols="20" rows="5">
   Sie k&ouml;nnen dies hier &uuml;berschreiben!
</textarea>
```

Angaben zu Höhe und Breite (Zeilen- und Spaltenzahl) sind obligatorisch und müssen vorgenommen werden. Um die Auswertung der Formulardaten zu erleichtern, sollte auch ein `name`-Attribut vergeben werden:

```
<textarea cols="20" rows="5" name="kommentar">
   Sie k&ouml;nnen dies hier &uuml;berschreiben!
</textarea>
```

Hinweis: Das `<textarea>`-Element kann mit CSS beliebig gestylt werden, wobei sämtliche CSS-Box-Properties (Border, Margin, Padding, Hintergrundfarbe, -bild etc.) angewendet werden können. Das Element verhält sich in diesem Zusammenhang wie ein »Inline-Block«. **[«]**

```
<p>Text davor <textarea cols="20" rows="2"
 style="border: 1px solid #aaa; margin: 10px; padding: 10px;
 background: url('bg.gif');">
   &Auml;nderungen m&ouml;glich.
</textarea> und danach.</p>
```

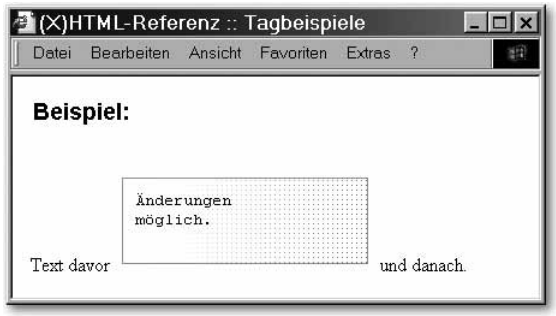

Abbildung 14.90 Textarea mit Hintergrundbild, Margin und Padding

[zB] *Beispiel:*

Eine Textarea ohne weitere Angaben (in Abbildung 14.91 oben) erhält vom Browser zwar Defaultabmessungen, ist aber weder in HTML noch in XHTML legal, da `cols`- und `rows`-Attribute vorgeschrieben (`#required`) sind. Für ein funktionstüchtiges Formular sollte zudem (unter anderem) jeweils ein `name`-Attribute vergeben werden.

```
<form>
<textarea></textarea>
<br>
<textarea cols="20" rows="2">Defaulttext, kann &uuml;berschrieben
werden.</textarea>
<br>
<textarea cols="35" rows="5">Lorem ipsum ...</textarea>
</form>
```

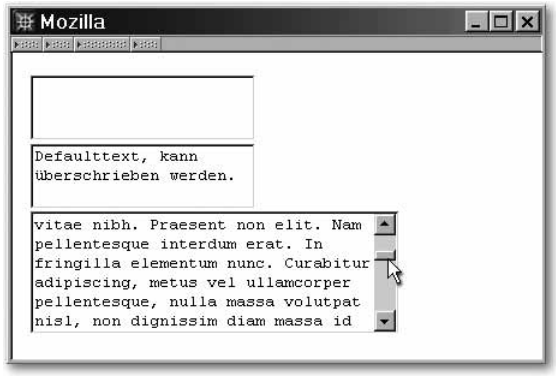

Abbildung 14.91 Textfelder können Defaulttext enthalten. Bei längeren Eingaben werden Scrollbalken eingeblendet.

14.83 tfoot

Logisches Element: Deklariert die Tabellenzeile(n), die der Container enthält, als Teil des Tabellenfußes.

14.83.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Container
XHTML-Modul	Tables
Kontext	Im `<table>`-Container
Erlaubte Inhalte	Tabellenzeilencontainer `<tr>`
Offizielle W3C Spezifikationen	HTML 4.0, 4.01; XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<tfoot> ... </tfoot>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
tfoot       { display: table-footer-group; }
```

14.83.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `char`, `charoff`, `valign`
Internet Explorer	`bgcolor`, *Access-*, *Edit-Attribute*

14.83.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte aller Zellen des Tabellenfußbereichs. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet die Inhalte aller Zellen linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert die Inhalte aller Zellen innerhalb des Zellraumes.
- `align="right"`
  richtet die Inhalte aller Zellen rechtsbündig zum Zellenrand aus.
- `align="justify"`
  richtet die Inhalte aller Zellen blocksatzmäßig aus.
- `align="char"`
  richtet die Inhalte aller Zellen anhand eines Zeichens relativ zueinander aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann mittels des `char`-Attributs von `<tfoot>` bestimmt werden.

[!] *Achtung:* Dieser Attributwert wird noch nirgends unterstützt.

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem die Inhalte der Zellen des Tabellenfußbereichs `<tfoot>` durch `align="char"` gegeneinander ausgerichtet werden sollen. Die Intention ist, auf diese Weise eine Tabellierung zu ermögli-

chen. Der Defaultwert ist je nach Spracheinstellung mittels `lang`-Attribut o.ä. der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Dieses Beispiel richtet die Inhalte der Zellen am Doppelpunkt aus:

```
<tfoot align="char" char=":"> ... </tfoot>
```

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent vom linken Zellrand der Zellen des Tabellenfußbereiches für das mittels des `char`-Attributs von `<tfoot>` bezeichnete Ausrichtungszeichen. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung der Inhalte aller Zellen des Tabellenfußbereichs. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zellen.

- `valign="top"`
  richtet die Inhalte aller Zellen am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert die Inhalte aller Zellen vertikal im Zellraum.
- `valign="bottom"`
  richtet die Inhalte aller Zellen am unteren Zellenrand aus.
- `valign="baseline"`
  richtet die Inhalte aller Zellen nach einer je Zeile gemeinsamen Textbasislinie aus.

14.83.4 Proprietäre Attribute – Internet Explorer

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bgcolor` bestimmt die Hintergrundfarbe des Tabellenfußbereichs. In Abwesenheit des Attributs gilt die Hintergrundfarbe der Tabelle oder des Dokuments.

CSS-Alternative: `tfoot {background-color:#rrggbb;}`

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.83.5 Erläuterung zum Element

Der `<tfoot>`-Container dient der logischen Gruppierung derjenigen Tabellenzeilen, die als Tabellenfuß fungieren sollen. In HTML ist die Endmarke `</tfoot>` als *omittable* (auslassbar) definiert. Ebenso wie beim `<thead>`-Container sollte man sie jedoch grundsätzlich nicht entfallen lassen, da dies die Abgrenzung von der folgenden Tabellenkörpergruppe erschwert oder unmöglich macht[58]. In XHTML muss die Endmarke grundsätzlich gesetzt werden.

Pro Tabelle ist höchstens eine `<tfoot>`-Gruppe gestattet, wobei der Container *nach* einer möglichen Tabellenüberschrift `<caption>`, *nach* eventuellen Spalten- oder Spaltengruppendefinitionen mit `<col>` oder `<colgroup>` und nach einem optionalen Tabellenkopfbereich `<thead>` stehen muss. Der logische Tabellenfuß-

58 Laut Spezifikation muss in diesem Fall die Startmarke `<tbody>` gesetzt werden, die ansonsten auslassbar wäre.

bereich steht also stets vor dem eigentlichen Tabelleninhalt: dem <tfoot>-Container folgt implizit oder explizit der erste <tbody>-Container.

Die Quelltextplatzierung des Fußbereiches relativ am Anfang der Tabelle (nach Überschrift und Kopfbereich) soll es einem Browser ermöglichen, den Fußbereich zu rendern, ohne dass sämtliche Tabellendaten geladen sein müssen[59]. Dargestellt wird der Tabellenfuß jedoch stets am Ende der Tabelle.

Als Inhalt von <tfoot> sind ausschließlich Zeilencontainer <tr> zugelassen. Das Element besitzt von sich aus keine sichtbare Präsentation. Seine Begrenzung zur folgenden Tabellenzeilengruppe kann aber mittels des rules-Attribut des <table>-Containers sichtbar gemacht werden: <table rules="groups"> erzeugt durchgezogene horizontale Linien zwischen allen Zeilengruppen.

Eine gesonderte Präsentation mittels CSS ist in gewissem Rahmen möglich, wobei lediglich Hintergrund- und Ausrichtungseigenschaften wirksam sind, da das CSS-Boxmodell auf <tfoot> nicht anwendbar ist.

14.84 th

Logisches Element: Definiert eine Tabellenkopfzelle (*table header*) als Unterelement eines Tabellenzeilencontainers <tr>.

14.84.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Container
XHTML-Modul	Basic Tables, Tables
Kontext	In Tabellenzeilencontainer <tr>
Erlaubte Inhalte	Tabellen, Block- und Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	XHTML mobile profile *nicht enthalten in:* HTML 4.0 mobile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<th> ... </th>
```

59 Für eine längere Tabelle könnte der Fußbereich (so die Theorie) im Browserfenster sichtbar gehalten werden.

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
th          { display: table-cell;
              font-weight: bolder;
              text-align: center; }
```

14.84.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onafterupdate`, `onbeforecopy`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `onbeforeupdate`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfilterchange`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onrowenter`, `onrowexit`, `onselectstart`

Elementspezifische Attribute

W3C	`abbr`, `align`, `axis`, `bgcolor`, `char`, `charoff`, `colspan`, `height`, `nowrap`, `rowspan`, `scope`, `valign`, `width`
Internet Explorer	`background`, `bordercolor`, `bordercolordark`, `bordercolorlight` *Access-Attribute, Databinding-Attribute, Edit-Attribute*
Netscape	`background`

14.84.3 Offizielle Attribute – W3C

abbr

Wert	*cdata*
Typ	Eine beliebige Zeichenkette (CDATA)
Verwendung	Optional

Barrierefreiheit: Das `abbr`-Attribut dient dem Hinzufügen einer kurzen Beschreibung des Inhaltes der Tabellenheaderzelle oder dessen Bedeutung, die beispielsweise von Screenreadern bei der Ausgabe dem Zelleninhalt vorangestellt werden kann.

Anmerkung: Der Attributwert kann von visuell orientierten User Agents auch stellvertretend für den Zelleninhalt verwendet werden, wenn Platzgründe dies erfordern.

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt die horizontale Ausrichtung des Inhalts der Tabellenheaderzelle. Defaultwert ist `"center"`.

- `align="left"`
  richtet den Inhalte der Zelle linksbündig zum Zellenrand aus.
- `align="center"` (Default)
  zentriert den Inhalt der Zelle innerhalb des Zellraumes.
- `align="right"`
  richtet den Inhalt der Zelle rechtsbündig zum Zellenrand aus.
- `align="justify"`
  richtet den Inhalt der Zelle blocksatzmäßig aus.
- `align="char"`
  richtet die Inhalte der Zelle anhand eines Zeichens aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann mittels des `char`-Attributs von `<th>` bestimmt werden.

  Achtung: Dieser Attributwert wird derzeit nirgends unterstützt. **[!]**

axis

Wert	*"kategorie_a [, kategorie_b, ..., kategorie_n]"*
Typ	Eine kommagetrennte Liste aus Kategoriebezeichnern
Verwendung	Optional

Barrierefreiheit: Das `axis`-Attribut ordnet die Tabellenheaderzelle einer oder mehreren Inhaltskategorien zu, um es dem User Agent zu ermöglichen, Inhalte zu klassifizieren oder (im Rahmen einer Query) für bestimmte Ausgaben zu fil-

tern. Eine typische Verwendung wäre der Einsatz der zugeordneten Kategorie(n) in einem Screenreader als Verschlagwortung des Zelleninhalts (vergleichbar mit dem `abbr`-Attribut).

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `bgcolor` bestimmt die Hintergrundfarbe der Tabellenzelle. In Abwesenheit des Attributs gilt die Hintergrundfarbe der umgebenden Tabellenzeile `<tr>`, der Tabelle oder des Dokuments.

Im *strict*-Vokabular ist das Attribut `bgcolor` nicht enthalten.

CSS-Alternative: `th {background-color:#rrggbb;}`

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem der Inhalt der Tabellenheaderzelle durch `align="char"` ausgerichtet werden soll. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Der Defaultwert ist je nach Spracheinstellung über das `lang`-Attribut o.ä. der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Dieses Beispiel richtet die Inhalte der Zelle am Doppelpunkt aus:

```
<th align="char" char=":"> ... </th>
```

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent vom linken Zellrand für das mittels des `char`-Attributs des Elements bezeichneten Ausrichtungszeichen. Auf diese Weise wird eine genaue Positionierung der Tabellierung

ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

colspan

Wert	*number*
Typ	Positive Ganzzahl größer 1
Verwendung	Optional

Das `colspan`-Attribut (*column span*) dehnt den Inhaltsbereich der Tabellenzelle nach rechts über mehrere Tabellenspalten aus. Die Zahl der überspannten Spalten wird durch eine positive Ganzzahl angegeben, wobei die Ursprungsspalte mitgezählt wird. Die `colspan`-Zelle steht daher in ihrer Zeile für eine entsprechende Zahl von Zellendefinitionen.

Hinweis: Übersteigt die Zahl der Zellendefinitionen durch ein `colspan` die Anzahl der real existierenden Zellen in anderen Zeilen, so wird dies dort durch Anfügen von Blindzellen am rechten Tabellenrand aufgefüllt. **[«]**

```
<table>
<tr>
<!-- drei Zellendefinitionen nach rechts durch colspan: -->
<th colspan="3">B&uuml;cher von Thomas Mann</th>
</tr>
<tr>
<!-- drei reale Zellendefinitionen: -->
<td>Tod in Venedig</td>
<td>Buddenbrooks</td>
<td>Zauberberg, Der</td>
</tr>
</table>
```

Anmerkung: Eine Zelle kann sowohl `colspan`- als auch `rowspan`-Attribut besitzen und sich daher gleichzeitig über mehrere Spalten wie Zeilen erstrecken.

height

Wert	*integerwert* | *prozentwert%*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `height`-Attribut bestimmt die Mindesthöhe der Tabellenheaderzelle als Pixelwert (Ganzzahl) oder mit einer Prozentan-

gabe. In Abwesenheit des Attributs wird die Höhe durch das `height`-Attribut der Tabellenzeile `<tr>` oder die Inhalte der Zelle bestimmt.

Anmerkung: Die Höhenangabe ist nur relevant, solange die Zelleninhalte und diejenigen der Nachbarzellen der gleichen Zeile dies zulassen: Die Zelle kann je nach den Abmessungen der enthaltenden Elemente (beispielsweise Grafiken) auch auf eine die Angaben überschreitende Höhe aufgespannt werden.

nowrap

Wert	*"nowrap"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `nowrap`-Attribut erlaubt bzw. unterdrückt den automatischen Zeilenumbruch innerhalb der Tabellenzelle. Defaultmäßig bricht der Browser lange Zeilen automatisch um. Der Wert `nowrap="nowrap"` verbietet dies. Gegebenenfalls erscheint durch die erhöhte Tabellenbreite eine horizontale Scrollleiste.

Anmerkung: Der Internet Explorer interpretiert zusätzlich die Werte `"true"` und `"false"`, wobei `nowrap="false"` den Umbruch explizit zulässt, während `nowrap="true"` ihn verbietet. Andere Browser unterstützen diese Werte nicht.

rowspan

Wert	*number*
Typ	Positive Ganzzahl größer 1
Verwendung	Optional

Das `rowspan`-Attribut dehnt den Inhaltsbereich der Tabellenzelle nach unten über mehrere Tabellenzeilen aus. Die Zahl der überspannten Zeilen wird durch eine positive Ganzzahl angegeben, wobei die Ursprungszeile mitgezählt wird. Die `rowspan`-Zelle steht daher in ihrer Spalte für eine entsprechende Zahl von Zellendefinitionen. Die realen Zellen der Folgezeile weichen den `rowspan`-Definitionen nach rechts aus.

[»] *Hinweis:* Übersteigt die Zahl der Zellendefinitionen durch ein `rowspan` die Anzahl der real existierenden Zellen in anderen Zeilen, so wird dies dort durch Anfügen von Blindzellen am rechten Tabellenrand aufgefüllt.

Beispiel: **[zB]**

```
<table>
<tr>
<!-- drei Zellendefinitionen nach unten durch rowspan: -->
<th rowspan="3">B&uuml;cher von Thomas Mann</th>
<td>Tod in Venedig</td>
</tr>
<tr>
<!-- hier gilt der rowspan aus Zeile 1-->
<td>Zauberberg, Der</td>
</tr>
<tr>
<!-- hier gilt der rowspan aus Zeile 1-->
<td>Zauberberg, Der</td>
</tr>
<tr>
<!-- ab hier rowspan beendet-->
<td> ... </td>
<td> ... </td>
</tr>
</table>
```

Anmerkung: Eine Zelle kann sowohl `colspan`- als auch `rowspan`-Attribut besitzen und sich daher gleichzeitig über mehrere Spalten wie Zeilen erstrecken.

scope

Wert	*"row"* | *"col"* | *"rowgroup"* | *"colgroup"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Accessibility: Das `scope`-Attribut nennt den Bezug der Tabellenheaderzelle zu einem Bereich von Tabellendatenzellen. Im einfachsten Fall ist dieser Bezug spalten- oder zeilenorientiert und erstreckt sich auf die betreffende Spalte bzw. Zeile, in der sich die Tabellenheaderzelle befindet. Ein Screenreader setzt beim Verlesen entsprechender Tabelleninhalte jeweils den Inhalt der zugeordneten Tabellenheaderzelle voran.

- `scope="col"` (Default)
  Die Tabellenheaderzelle nimmt Bezug auf ihre Tabellenspalte.
- `scope="row"`
  Die Tabellenheaderzelle nimmt Bezug auf ihre Tabellenzeile.

- `scope="colgroup"`
  Die Tabellenheaderzelle nimmt Bezug auf eine mit `<colgroup>` definierte Spaltengruppe.
- `scope="rowgroup"`
  Die Tabellenheaderzelle nimmt Bezug auf eine Zeilengruppe, wie sie mit `<thead>`, `<tbody>` oder `<tfoot>` definiert werden kann.

[zB] *Beispiel:*

Die Headerzellen sind mit `scope="col"` ihren Spalten als Spaltenüberschriften zugeordnet:

```
<table>
 <tr>
   <th scope="col">Kopfdaten Spalte 1</th>
   <th scope="col">Kopfdaten Spalte 2</th>
 </tr>
 <tr>
   <td>Zelleninhalt, Feld 1 Spalte 1</td>
   <td>Zelleninhalt, Feld 1 Spalte 2</td>
 </tr>
 ...
</table>
```

Die Headerzellen sind mit `scope="row"` ihren Zeilen zugeordnet:

```
<table>
 <tr>
   <th scope="row">Kopfdaten Zeile 1</th>
   <td>Zelleninhalt, Feld 1 Zeile 1</td>
   ...
 </tr>
 <tr>
   <th scope="row">Kopfdaten Zeile 2</th>
   <td>Zelleninhalt, Feld 1 Spalte 2</td>
   ...
 </tr>
</table>
```

Die zweite Headerzelle ist allen Datenzellen der Spaltengruppe zugeordnet:

```
<table>
<colgroup/><colgroup span ="3"/>
 <tr>
   <th scope="col">Land</th>
   <th scope="colgroup" colspan="3">Sorte</th>
 </tr>
```

```
  <tr>
    <td>Holland</td>
    <td>Rosen</td>
    <td>Tulpen</td>
    <td>Nelken</td>
  </tr>
  ...
</table>
```

Die Headerzelle ist allen Datenzellen der Zeilengruppe `<tbody>` zugeordnet:

```
<table>
<tbody>
 <tr>
   <th scope="rowgroup" colspan="3">Blumen aus Holland</th>
</tr>
</tr>
   <td>Rosen</td>
   <td>Tulpen</td>
   <td>Nelken</td>
 </tr>
</tbody>
 ...
</table>
```

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung des Inhalts der Tabellenheaderzelle. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zelle.

- `valign="top"`
  richtet den Inhalt der Zelle am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert den Inhalt der Zelle vertikal im Zellraum.
- `valign="bottom"`
  richtet den Inhalt der Zelle am unteren Zellenrand aus.
- `valign="baseline"`
  richtet die Inhalte aller Zellen der Zeile nach einer gemeinsamen Textbasislinie aus. Dieser Attributwert wirkt zeilenorientiert, sobald in einer Zeile eine Zelle mit diesem Wert auftritt.

width

Wert	*integerwert* | *prozentwert%*
Typ	Ganzzahl oder Ganzzahl mit folgendem Prozentzeichen
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `width`-Attribut bestimmt die Mindestbreite der Tabellenheaderzelle als Pixelwert (Ganzzahl) oder mit einer Prozentangabe. In Abwesenheit des Attributs wird die Breite durch die Breite der Inhalte der Zelle selbst oder der breitesten Zelle ihrer Spalte bestimmt.

Die Breitenabgabe ist nur relevant, solange die Zelleninhalte dies zulassen: Die Zelle kann je nach den Abmessungen der sie enthaltenden Elemente (beispielsweise Grafiken) auch auf eine die Angaben überschreitende Breite aufgespannt werden.

14.84.4 Proprietäre Attribute – Internet Explorer

Internet Explorer

background

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `background` nennt den URL einer Hintergrundgrafik für die Tabellenzelle (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich.

Übliche Grafikformate sind JPG und GIF, wobei auch transparente und (browserabhängig) animierte GIFs gestattet sind. Eine gleichzeitig mit `bgcolor` definierte Hintergrundfarbe gilt als *hinter* der Hintergrundgrafik befindlich – sie ist durch eventuelle transparente Bereiche einer Hintergrundgrafik sichtbar.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt die einheitliche Rahmenfarbe der Tabellenzelle. In Abwesenheit des Attributs gilt die Rahmenfarbe der umgebenden Tabellenzeile `<tr>`, diejenige der Tabelle oder die Defaultfarbe des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

bordercolordark

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolordark` bestimmt die dunkle obere Rahmenfarbe (Schattenfarbe) der Tabellenzelle. In Abwesenheit des Attributs gilt der Wert des `bordercolor`-Attributs, die Rahmenfarbe der umgebenden Tabellenzeile `<tr>`, diejenige der Tabelle oder die Defaultfarbe des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

bordercolorlight

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolorlight` bestimmt die helle untere Rahmenfarbe (Lichtfarbe) der Tabellenzelle. In Abwesenheit des Attributs gilt der Wert des `bordercolor`-Attributs, die Rahmenfarbe der umgebenden Tabellenzeile `<tr>`, diejenige der Tabelle oder die Defaultfarbe des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.84.5 Proprietäre Attribute – Netscape

background

Wert	*URL*
Typ	Ein URI-String
Verwendung	Optional

Das Attribut `background` nennt den URL einer Hintergrundgrafik für die Tabellenzelle (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich.

14.84.6 Erläuterung zum Element

Der `<th>`-Container bezeichnet eine Tabellendatenzelle mit Headerfunktion (Kopfzelle), beschreibt ihren Inhalt also als einer Spalte oder Zeile übergeordnet. Die Endmarke `</th>` ist in HTML als *omittable* (auslassbar) definiert. In XHTML muss sie hingegen grundsätzlich gesetzt werden. Das Element tritt ausschließlich als Kindelement des Tabellenzeilen-Containers `<tr>` auf.

Gewöhnlich wird eine Headerzelle innerhalb der ersten Tabellenzeile in Form einer Spaltenüberschrift eingesetzt, wobei sie ihren Inhalt fett und zentriert präsentiert. Die entsprechende Tabellenzeile kann als zusätzliche Verdeutlichung in ein `<thead>`-Element eingeschlossen sein.

Der Zelleninhalt soll von einem User Agent als Spaltenüberschrift interpretiert werden, wobei alternativ auch der Inhalt des `abbr`-Attribut (sofern gesetzt) herangezogen werden kann.

Soll die Kopfzelle sich nicht auf eine Spalte beziehen, was defaultmäßig der Fall ist, so kann dies mit dem `scope`-Attribut veranlasst werden. Durch Setzen von `scope="row"` wird die Kopfzelle im Zeilenkontext interpretiert.

Weitere Beziehungen zwischen Daten- und Headerzellen sind über eine `ID-IDREFS`-Beziehung herstellbar, wobei ein `header`-Attribut einer Datenzelle `<td>` auf das jeweilige `id`-Attribut einer oder mehrerer Headerzellen `<th>` verweisen kann.

Durch CSS kann eine Kopfzelle wie eine gewöhnliche Tabellenzelle präsentiert werden. Neben der Formatierung des Zellinhalts sind CSS-Anweisungen für Hintergrund und Zellrahmen einsetzbar, nicht jedoch solche für Padding und Margin. Die HTML-Defaultpräsentation (»fett«, »zentriert«) kann durch CSS-Anwei-

sungen überschrieben werden, wodurch die Kopfzelle optisch wie eine reguläre Datenzelle präsentiert werden kann, ohne ihren Status einzubüßen.

Hinweise zur Barrierefreiheit

Verwenden Sie für Datentabellen die entsprechenden Attribute `scope` und `axis`, um die Beziehung zwischen Kopf- und Datenzellen zu verdeutlichen oder um weitere inhaltliche Informationen zu bieten. Vergeben Sie für `<th>` ein `id`-Attribut, wenn Sie von einer Datenzelle aus einen Bezug auf eine Headerzelle bilden wollen. Stellen Sie für umfangreiche Zelleninhalte mit Hilfe des `abbr`-Attributs eine Abkürzung bereit. Setzen Sie auch das `summary`-Attribut für `<table>` ein, um eine Beschreibung der Tabelle zur Verfügung zu stellen. (Die Ansprüche an barrierefreie Tabellen sind in der BITV, Abschnitt 5.1, beschrieben.)

14.85 thead

Logisches Element: Das `<thead>`-Element deklariert die Tabellenzeile(n), die der Container enthält, als Teil des Tabellenkopfs.

14.85.1 Elementeigenschaften

Einführung	HTML 4.0
Elementtyp	Container
XHTML-Modul	Tables
Kontext	Im Tabellencontainer `<table>`
Erlaubte Inhalte	Tabellenzeilencontainer `<tr>`
Offizielle W3C Spezifikationen	HTML 4.0, 4.01 XHTML 1.0, 1.1, 2.0;
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<thead> ... </thead>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
thead       { display: table-header-group; }
```

14.85.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `char`, `charoff`, valign
Internet Explorer	`bgcolor`, *Access-, Edit-Attribute*

14.85.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte aller Zellen des Tabellenkopfbereichs. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet die Inhalte aller Zellen linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert die Inhalte aller Zellen innerhalb des Zellraumes.
- `align="right"`
  richtet die Inhalte aller Zellen rechtsbündig zum Zellenrand aus.
- `align="justify"`
  richtet die Inhalte aller Zellen blocksatzmäßig aus.

- `align="char"`
  richtet die Inhalte aller Zellen anhand eines Zeichens relativ zueinander aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann mittels des `char`-Attributs von `<thead>` bestimmt werden.

  Achtung: Dieser Attributwert wird derzeit nirgends unterstützt. **[!]**

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem die Inhalte der Zellen des Tabellenkopfbereichs `<thead>` durch `align="char"` gegeneinander ausgerichtet werden sollen. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Der Defaultwert ist je nach Spracheinstellung über `lang`-Attribut o.ä. der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent, dabei wird vom linken Zellrand der Zellen des Tabellenkopfbereichs für das über das `char`-Attributs von `<thead>` bezeichnete Ausrichtungszeichen gerechnet. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

valign

Wert	*"top"* | *"middle"* | *"bottom"* | *"baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung der Inhalte aller Zellen des Tabellenkopfbereichs. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zellen.

- `valign="top"`
  richtet die Inhalte aller Zellen am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert die Inhalte aller Zellen vertikal im Zellraum.
- `valign="bottom"`
  richtet die Inhalte aller Zellen am unteren Zellenrand aus.
- `valign="baseline"`
  richtet die Inhalte aller Zellen nach einer je Zeile gemeinsamen Textbasislinie aus.

14.85.4 Proprietäre Attribute – Internet Explorer

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bgcolor` bestimmt die Hintergrundfarbe des Tabellenkopfbereichs. In Abwesenheit des Attributs gilt die Hintergrundfarbe der Tabelle oder des Dokuments.

CSS-Alternative: `thead {background-color:#rrggbb;}`

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.85.5 Erläuterung zum Element

Das Element `<thead>` gruppiert mehrere Tabellenzeilen zu einem logischen Bereich, der als Tabellenkopfgruppe dient. Pro Tabelle ist höchstens eine

<thead>-Gruppe gestattet, wobei der Container *nach* einer möglichen Tabellenüberschrift <caption> und *nach* eventuellen Spalten- oder Spaltengruppendefinitionen mit <col> oder <colgroup> stehen muss. Dem <thead>-Container folgt implizit oder explizit ein <tbody>-Container. (»Freie« Tabellenzeilen, die keiner Gruppe zugeordnet sind, können dem <thead>-Bereich nicht vorangehen – ist eine Headergruppe definiert, so muss die Tabelle mit ihm beginnen.)

In HTML ist die Endmarke </thead> als *omittable* (auslassbar) definiert. Sie sollte allerdings grundsätzlich nicht entfallen, da dies die Abgrenzung von der folgenden Tabellenkörpergruppe erschwert oder unmöglich macht[60]. In XHTML muss die Endmarke grundsätzlich gesetzt werden.

Als Inhalt von <thead> sind ausschließlich Zeilencontainer <tr> zugelassen. Die Möglichkeit, deren Inhalt in diesem Kontext auf <th>-Elemente zu beschränken, besteht dabei nicht. Die Tabellenzeilen eines <thead>-Bereichs können daher beliebig <th>- und <td>-Container enthalten.

Das Element besitzt von sich aus keine sichtbare Präsentation. Seine Begrenzung zur folgenden Tabellenzeilengruppe kann aber über das rules-Attribut des <table>-Containers sichtbar gemacht werden: <table rules="groups"> erzeugt durchgezogene horizontale Linien zwischen allen Zeilengruppen.

Eine gesonderte Präsentation mittels CSS ist in gewissem Rahmen möglich, wobei lediglich Hintergrund- und Ausrichtungseigenschaften wirksam sind, da das CSS-Boxmodell auf <thead> nicht anwendbar ist. Es ist jedoch möglich[61], mit height eine Höhe des Tabellenkopfbereiches anzugeben, wenn gleichzeitig das overflow-Property mit dem Wert "auto" oder "scroll" gesetzt wird (die Werte "visible" oder "hidden" sind nicht anwendbar).

Beispiel: **[zB]**

```
<table rules="groups">
<caption>Tabelle mit Kopfgruppe</caption>
   <thead>
      <tr><th>Spalte 1</th><th>Spalte 2</th></tr>
   </thead>
   <tbody>
      <tr><td>Wert 1</td><td>Wert 2</td></tr>
   </tbody>
</table>
```

60 Laut Spezifikation muss in diesem Fall die Startmarke <tbody> gesetzt werden, die ansonsten auslassbar wäre.

61 Derzeit nur in Mozilla/Firefox.

14.86 title

Strukturelement. Das `<title>`-Enthält den Dokumenttitel und ist obligatorischer Bestandteil des Dokumentkopfes `<head>`.

14.86.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Strukturcontainer
XHTML-Modul	Structure
Kontext	Obligatorischer Teil des Dokumentkopfs `<head>`
Erlaubte Inhalte	`#CDATA` (Text)
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<title> ... </title>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
-
```

14.86.2 Attribute (Übersicht)

Standardattribute

HTML	`dir`, `lang`
XHTML	`id`, `dir`, `xml:lang`

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	onreadystatechange

Elementspezifische Attribute

W3C	–

14.86.3 Erläuterung zum Element

Das Strukturelement `<title>` definiert den Dokumenttitel des HTML-Dokuments und ist als solcher obligatorischer Bestandteil des durch den `<head>`-Containers (siehe 1) definierten Dokumentkopfes. Der Dokumenttitel erscheint nicht im Browserfenster selbst, sondern in der Titelzeile des Anzeigefensters. Der Inhalt des `<title>`-Containers zählt daher nicht zum Textinhalt des Dokuments.

Suchmaschinen betrachten den Dokumenttitel als vorrangig in der Inhaltsbeschreibung des Dokuments und verwenden ihn bei der Generierung von Suchergebnisseiten als Linktext und um einen Verweis auf die Seite zu bilden. Ebenso dient der Titel in der Liste besuchter Seiten und beim Setzen eines Bookmarks als Text für den jeweiligen Eintrag.

Der Elementinhalt von `<title>` ist in den DTDs von HTML und XHTML mit `#PCDATA` definiert, was den Einsatz von Entitäten und deren Ersetzung erlaubt. Elementinhalt darf hingegen nicht auftreten bzw. wird nicht als solcher geparst, da die Tagklammern nicht als Steuerzeichen gewertet werden.

Die HTML-DTD schließt dennoch Elementinhalt bestehend aus den restlichen, im Dokumentkopf erlaubten Elementen sicherheitshalber explizit aus:

```
<!ELEMENT TITLE - - (#PCDATA) -(%head.misc;)
 -- document title -->
```

Hierbei beschreibt das Parameterentity `%head.misc` die Elemente `<script>`, `<style>`, `<meta>`, `<link>` und `<object>`. Dies ist in XHTML nicht formulierbar[62], weshalb die DTD-Definition hier lediglich lautet:

```
<!ELEMENT title (#PCDATA)>
```

Aufgrund der Inhaltsdefinition mit `#PCDATA` darf der Browser eine **Weißraumnormalisierung** vornehmen, also führende und folgende Leerzeichen entfernen sowie mehrere einander folgende Weißraumzeichen zu einem Leerzeichen zusammenfassen. Der Textinhalt des `<title>`-Containers darf sich also ohne Weiteres über mehrere Quelltextzeilen erstrecken:

```
<title>   Das
     ist
  der Dokumenttitel    </title>
```

62 XML erlaubt keine Ausschlüsse in der Definition von Elementinhalten.

ist von der Anzeige identisch mit

```
<title>Das ist der Dokumenttitel</title>
```

[»] *Hinweis:* Eine Formatierung des Inhalts des `<title>`-Containers mit CSS ist nicht möglich. Geschützte Leerzeichen ` ` werden jedoch ausgewertet und können in geringem Maße zur Gestaltung eingesetzt werden (Einrückung am Anfang des Titels, größere Wortabstände).

Zugriff auf <title> mittels DOM

Mittels des DOM-Properties `document.title` ist die Zeichenkette, die der `<title>`-Container enthält, direkt auslesbar.

14.87 tr

Das Element `<tr>` definiert eine Tabellenzeile (*table row*) als Unterelement von `<table>` oder einer Tabellenzeilengruppe `<thead>`, `<tbody>` oder `<tfoot>`.

14.87.1 Elementeigenschaften

Einführung	HTML 3.2
Elementtyp	Container
XHTML-Modul	Basic Tables, Tables
Kontext	In `<table>`, `<thead>`, `<tbody>`, `<tfoot>`
Erlaubte Inhalte	Tabellenzellencontainer `<td>` oder `<th>`
Offizielle W3C Spezifikationen	HTML 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	XHTML mobile profile *nicht enthalten in:* HTML 4.0 mobile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<tr> ... </tr>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
tr { display:table-row; }
```

14.87.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforeeditfocus`, `onbeforepaste`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onerrorupdate`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`align`, `bgcolor`, `char`, `charoff`, `valign`
Internet Explorer	`background`, `bordercolor`, `bordercolordark`, `bordercolorlight` *Access-, Edit-Attribute*

14.87.3 Offizielle Attribute – W3C

align

Wert	*"left"* | *"center"* | *"right"* | *"justify"* | *"char"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `align`-Attribut bestimmt die horizontale Ausrichtung der Inhalte aller Zellen der Tabellenzeile. Defaultwert ist `"left"`.

- `align="left"` (Default)
  richtet die Inhalte aller Zellen linksbündig zum Zellenrand aus.
- `align="center"`
  zentriert die Inhalte aller Zellen innerhalb des Zellraumes.
- `align="right"`
  richtet die Inhalte aller Zellen rechtsbündig zum Zellenrand aus.

- `align="justify"`
  richtet die Inhalte aller Zellen blocksatzmäßig aus.
- `align="char"`
  richtet die Inhalte aller Zellen der Zeile anhand eines Zeichens aus. Dies kann beispielsweise ein Dezimalpunkt oder Komma sein. Das gewünschte Zeichen kann mittels des `char`-Attributs von `<tr>` bestimmt werden.

[!] *Achtung:* Dieser Attributwert wird derzeit nirgends unterstützt.

bgcolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Deprecated (Präsentationsattribut): Das Attribut `bgcolor` bestimmt die Hintergrundfarbe der Tabellenzeile, wobei das Tabellengitter, sofern sichtbar, nicht gefärbt wird. In Abwesenheit des Attributs gilt die Hintergrundfarbe der Tabelle oder des Dokuments.

Im *strict*-Vokabular ist das Attribut `bgcolor` nicht enthalten.

CSS-Alternative: `tr {background-color:#rrggbb;}`

[zB] *Beispiel:*

```
<table width="300" border="1">
<tr><th>Spalte 1</th><th>Spalte 2</th><th>Spalte 3</th></tr>
<tr bgcolor="#aaaaaa">
  <td>Wert 1-1</td><td>Wert 1-2</td><td>Wert 1-3</td>
</tr>
  ...
</table>
```

Abbildung 14.92 Tabellenzeile mit Hintergrundfarbe

char

Wert	*character*
Typ	Einzelnes ASCII-Zeichen
Verwendung	Optional

Das `char`-Attribut bestimmt das Zeichen, nach dem die Inhalte der Zellen der Tabellenzeile durch `align="char"` ausgerichtet werden sollen. Die Intention ist, auf diese Weise eine Tabellierung zu ermöglichen. Der Defaultwert ist je nach Spracheinstellung über das `lang`-Attribut o.ä. der Dezimalpunkt (`lang="en"`) oder das Komma (z. B. bei `lang="de"`).

Dieses Beispiel richtet die Inhalte der Zellen am Doppelpunkt aus:

```
<tr align="char" char=":"> ... </tr>
```

charoff

Wert	*number[%]*
Typ	Eine Ganzzahl als Pixelwert oder eine Prozentangabe
Verwendung	Optional

Das `charoff`-Attribut bestimmt einen Offset in Pixel oder Prozent vom linken Zellrand der Zellen der Tabellenzeile für das Ausrichtungszeichen, das über das `char`-Attribut von `<tr>` bezeichnet wird. Auf diese Weise wird eine genaue Positionierung der Tabellierung ermöglicht. Ist das Tabellierungszeichen im Zellinhalt nicht enthalten, so wird das Ende des Inhalts an der `charoff`-Position ausgerichtet.

valign

Wert	*"top" | "middle" | "bottom" | "baseline"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Das `valign`-Attribut bestimmt die vertikale Ausrichtung der Inhalte aller Zellen der Tabellenzeile. Defaultwert ist `"middle"`, also die vertikale Zentrierung der Inhalte innerhalb der Zellen.

- `valign="top"`
  richtet die Inhalte aller Zellen der Zeile am oberen Zellenrand aus.
- `valign="middle"` (Default)
  zentriert die Inhalte aller Zellen der Zeile vertikal im Zellraum.
- `valign="bottom"`
  richtet die Inhalte aller Zellen der Zeile am unteren Zellenrand aus.

- `valign="baseline"`
  richtet die Inhalte aller Zellen der Zeile nach einer gemeinsamen Textbasislinie aus.

14.87.4 Proprietäre Attribute – Internet Explorer

background

Wert	*URL*
Typ	Der URL einer Grafik, die als Hintergrund dienen soll
Verwendung	Optional

Das Attribut `background` nennt den URL einer Hintergrundgrafik für die Tabellenzeile (Pfad/Dateiname). Sofern die Grafik von den Abmessungen nicht formatfüllend ist, wird sie entsprechend horizontal und vertikal gekachelt (wiederholt). Eine Skalierung der Grafik ist nicht möglich.

Übliche Grafikformate sind JPG und GIF, wobei auch transparente und (browserabhängig) animierte GIFs gestattet sind. Eine gleichzeitig mit `bgcolor` definierte Hintergrundfarbe gilt als *hinter* der Hintergrundgrafik befindlich – sie ist durch eventuelle transparente Bereiche einer Hintergrundgrafik sichtbar.

bordercolor

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolor` bestimmt die einheitliche Rahmenfarbe der Tabellenzeile. In Abwesenheit des Attributs gilt die Rahmenfarbe der Tabelle oder die Defaultfarbe des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

bordercolordark

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolordark` bestimmt die dunkle obere Rahmenfarbe (Schattenfarbe) der Tabellenzeile. In Abwesenheit des Attributs gilt der Wert des `bordercolor`-Attributs, die Rahmenfarbe der Tabelle oder die Default-Schattenfarbe (meist `#8C8E8C`) des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

bordercolorlight

Wert	*#rrggbb* | *farbbezeichner*
Typ	Farbname oder RGB-Wert in Hexadezimalschreibweise
Verwendung	Optional

Das Attribut `bordercolorlight` bestimmt die helle untere Rahmenfarbe (Lichtfarbe) der Tabellenzeile. In Abwesenheit des Attributs gilt der Wert des `bordercolor`-Attributs, die Rahmenfarbe der der Tabelle oder die Default-Lichtfarbe (meist `#C6C3C6`) des Tabellenrands.

Anmerkung: Das Attribut wird in Mozilla/Firefox und Opera nicht unterstützt.

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.87.5 Erläuterung zum Element

Der Tabellenzeilencontainer `<tr>` umschließt die Datenzellencontainer einer Tabelle, die zusammen eine Tabellenzeile bilden sollen. In der HTML-DTD ist die Endmarke `</tr>` als *omittable* definiert, darf also entfallen. In XHTML dagegen muss der Container stets geschlossen werden.

Der Zeilencontainer kann direkt als Unterelement des Tabellencontainers `<table>` oder als Unterelement von Zeilengruppencontainern wie `<thead>`, `<tfoot>` oder `<tbody>` auftreten. Sobald eine Zeilengruppierung vorgenommen wird, dürfen in HTML jedoch keine `<tr>`-Elemente außerhalb von Gruppen erscheinen, also nicht mehr als direkte Kindelemente von `<table>`.[63]

63 In HTML wird implizit ein `<tbody>` um »freie« Tabellenzeilen ergänzt, wenn diese als Kindelemente von `<table>` auftreten. In XHTML sind »freie« `<tr>` nach `<thead>` oder `<tfoot>` gestattet, können jedoch nicht mit `<tbody>`-Gruppen gemischt auftreten.

Die Anzahl der in der Zeile enthaltenen Zelldefinitionen bestimmt die Zahl der Spalten der Tabellen. Bei den Zelldefinitionen kann es sich um normale Datenzellen `<td>` oder Headerzellen `<th>` handeln oder um Definitionen, die durch Column- oder Rowspans erzeugt sind. Andere Elemente sind als unmittelbare Kindelemente von `<tr>` nicht gestattet. Die Spaltenzahl je Zeile muss für die gesamte Tabelle konstant sein, um Darstellungsprobleme zu vermeiden.

Der `<tr>`-Container selbst ist unsichtbar, kann jedoch HTML- oder CSS-Eigenschaften wie Hintergrund- oder Ausrichtungsattribute zugewiesen bekommen, die an die enthaltenen Zellen weitervererbt werden. Es können jedoch keine Eigenschaften der CSS-Box eingesetzt werden, die nur auf Blockelemente angewendet werden können – die Properties `border`, `margin` und `padding` sind auf `<tr>`-Container nicht anwendbar. Der Versuch, `<tr>` durch `display:block` entsprechend umzudeuten, bricht die Tabellendarstellung auf. Daher besteht keine Möglichkeit, eine Höhe für die Tabellenzeile anzugeben – diese wird durch die Höhe der höchsten in ihr enthaltenen Datenzelle bestimmt.

[zB] *Beispiel:*

Tabellenzeilen unmittelbar in `<table>`:

```
<table>
   <tr>
      <td>1</td><td>2</td>
   </tr>
   ...
</table>
```

... und als Unterelement einer durch `<tbody>` gebildeten Zeilengruppe:

```
<table>
  <tbody>
   <tr>
      <td>1</td><td>2</td>
   </tr>
  </tbody>
</table>
```

14.88 tt

Präsentationselement: Das Element `<tt>` bewirkt die Darstellung des Elementinhalts mit nicht-proportionaler Schrift (*teletype*).

14.88.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Presentation
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1 **abgeschafft in:** XHTML 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<tt> ... </tt>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
tt      { display: inline;
          font-family: monospace; }
```

14.88.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.88.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.88.4 Erläuterung zum Element

Der `<tt>`-Container stellt seinen Inhalt in nicht-proportionaler Schrift dar, in der Anmutung der Ausgabe eines Fernschreibers (*teletypewriter*). Üblich sind Monospace-Fonts wie Courier, Courier New oder vergleichbare.

Bei dem Element handelt es sich zwar um ein Präsentationselement, jedoch zählt es nicht zu den vom W3C missbilligten Elementen: Es ist Teil des »Presentation«-Moduls von XHTML.

Als darstellungsgleiche Alternativen können (je nach Kontext) die Elemente `<code>`, `<samp>` oder `<kbd>` dienen, die ebenfalls eine Präsentation in Monospace-Font bewirken. Anders als `<tt>` zählen sie jedoch zu den logischen Elementen.

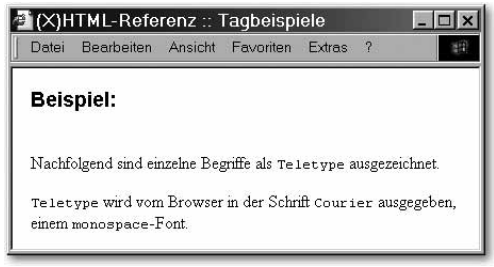

Abbildung 14.93 Präsentationsauszeichnung mit Teletype <tt>

[zB] *Beispiel:*

```
<p>Nachfolgend sind einzelne Begriffe als <tt>Teletype</tt>
ausgezeichnet.</p>
```

```
<p><tt>Teletype</tt> wird vom Browser in der Schrift
<tt>Courier</tt> ausgegeben, einem <tt>monospace</tt>-Font.</p>
```

14.89 u

Deprecated: Das Präsentationselement `<u>` (*underline*) stellt den eingeschlossenen Textinhalt im Sinne einer Hervorhebung mit Unterstrich dar.

14.89.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Legacy
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0 / 4.01 (tr/fs); XHTML 1.0 (tr/fs); **abgeschafft in:** HTML 4.0/4.01 strict XHTML 1.0 strict, 1.1, 2.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<u> ... </u>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
u     { display: inline;
        text-decoration: underline; }
```

14.89.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`

Intrinsic Events Collection	–
Internet Explorer Events	onbeforecopy, onbeforecut, onbeforepaste, onblur, oncontextmenu, oncopy, oncut, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onfocus, onhelp, onlosecapture, onpaste, onpropertychange, onreadystatechange, onresize, onselectstart

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.89.3 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.89.4 Erläuterung zum Element

Das Element <u> bewirkt eine Textunterstreichung und ist somit ein reines Präsentationselement. Als durch das W3C missbilligtes Element ist es nicht mehr Teil der »strict«-Tagvokabulare von HTML und XHTML, ist aber aus Gründen der Abwärtskompatibilität noch im Legacy-Modul von XHTML enthalten.

Im Allgemeinen sollte man wegen der Verwechselungsmöglichkeit mit dem ebenfalls unterstrichen präsentierten Hyperlink auf die Verwendung des <u>-Containers verzichten. In Textbrowsern wird das Element aus diesem Grund selten unterstützt. Screenreader dagegen können Unterstreichungen ansagen. Dies hängt von den User-Einstellungen ab (defaultmäßig wird der Unterstrich ignoriert). Ein kurzes Beispiel hierzu:

```
Hier ist ein <u>Wort</u> unterstrichen.
```

Ist die Option **Underline change** aktiviert, so verliest der Screenreader HAL (**www.yourdolphin.com**) den vorstehenden Quelltext in etwa als:

»Hier ist ein **UNDERLINED** Wort **NORMAL** unterstrichen« **[Pause]**

Ist die Option nicht gewählt, so ignoriert der Screenreader die Marke.

Beispiel: **[zB]**

```
<p><u>Dieser Text ist einfach nur unterstrichen.</u></p>

<p>Dieser Text <a href="#">ist ein Link</a>!</p>

<p>In einem
<a style="text-decoration: none;" accesskey="o" href="#">
Link <u>o</u>hne Unterstrich
</a> kann mit &lt;u&gt; der Accesskey hervorgehoben werden.</p>
```

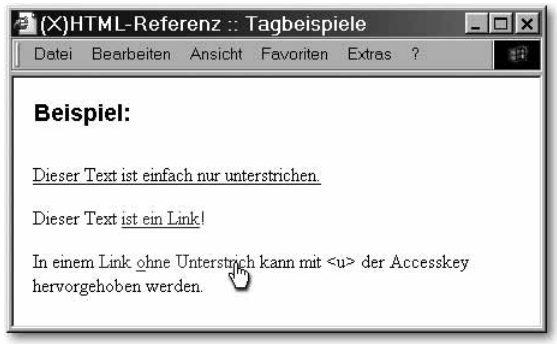

Abbildung 14.94 Die Verwendung von <u> ist nur selten sinnvoll.

14.90 ul

Logisches Element: Der Container `<ul>` (*unordered list*) erzeugt eine ungeordnete Liste, deren Listenelemente `<li>` durch Bullets gekennzeichnet werden.

14.90.1 Elementeigenschaften

Einführung	HTML 1.0
Elementtyp	Blockcontainer
XHTML-Modul	List
Kontext	Erlaubt innerhalb von `<applet>`, `<blockquote>`, `<body>`, `<button>`, `<center>`, `<dd>`, `<del>`, `<div>`, `<fieldset>`, `<form>`, `<iframe>`, `<ins>`, `<li>`, `<map>`, `<noframes>`, `<noscript>`, `<object>`, `<td>`, `<th>`

Erlaubte Inhalte	Listenelemente `<li>`
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0, 2.0, 3.2, 4.0, 4.01 XHTML 1.0, 1.1, 2.0; XHTML Basic
Mobil-Spezifikationen	cHTML, HTML 4.0 mobile XHTML mobile profile, iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<ul> ... </ul>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
ul      { margin-top: 1.33em;
          margin-right: 0px;
          margin-bottom: 1.33em;
          margin-left: 40px; }
```

... für verschachtelte Listen gilt jedoch:

```
ul  ul  { margin-top: 0px;
          margin-bottom: 0px; }
```

14.90.2 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecopy`, `onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncopy`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	`compact`, `type`
Internet Explorer	Access-, Edit-Attribute

14.90.3 Offizielle Attribute – W3C

compact

Wert	*"compact"*
Typ	Festgelegter Bezeichner oder leeres Marker-Attribut
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `compact`-Attribut reduziert den Abstand zwischen den Listenelementen der Liste und bewirkt eine kompaktere Darstellung der Liste.

Anmerkung: Das `compact`-Attribut wird in der Praxis von kaum einem Browser unterstützt. Es ist anzuraten, für die beabsichtigte Wirkung auf das CSS-`margin`-Property der Listenelemente zurückzugreifen.

type

Wert	*"disc"* | *"circle"* | *"square"*
Typ	Festgelegte Bezeichner
Verwendung	Optional

Deprecated (Präsentationsattribut): Das `type`-Attribut bestimmt die Art des Bullets der `<ul>`-Liste. Der übliche Defaultwert ist `"disc"`.

- `type="disc"`
  erzeugt ein Bullet in Form einer schwarzen Scheibe.
- `type="circle"`
  erzeugt ein Bullet in Form eines Kreises.
- `type="square"`
  erzeugt ein Bullet in Form eines schwarzen Quadrats.

Anmerkung: Der Wert des `type`-Attributs eines eingeschlossenen Listenelements `<li>` hat stets Vorrang vor der defaultmäßig oder per `type`-Attribut vorgegebenen Bulletform des umgebenden Listencontainers.

Im Falle einer ungeordneten Liste `<ul>` ist damit auch die Zuweisung eines Nummerierungszeichen (1, a, A, i, I) für ein oder mehrere enthaltene Listenelemente

möglich. Die vor dem Listenelement angezeigte Nummerierung ergibt sich aus dessen Position in der Liste, als ob es sich bei dieser um eine geordnete Liste handelte.

[!] *Achtung:* Im Internet Explorer setzt der Versuch, ein artfremdes Bullet festzulegen, das entsprechende Listenelement auf den *Defaulttyp* der umgebenden Liste (*nicht* etwa auf den dort per `type`-Attribut vorgegebenen Typ).

14.90.4 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.90.5 Erläuterung zum Element

Das `<ul>`-Element dient als Container für Listenelemente `<li>`, die eine so genannte »ungeordnete« Liste bilden sollen, also eine Liste aus nicht nummerierten Elementen. Als Aufzählungszeichen werden den Listenelementen neutrale Symbole wie Scheiben (das Defaultsymbol *disc*, gelegentlich auch in Form eines Rombusses oder hochkant gestellten Quadrats angezeigt), Quadrate (*square*) oder Kreise (*circle*) vorangestellt.

Alternativ ist per CSS neben der Deaktivierung des Aufzählungssymbols (`list-style-type:none`) auch der Einsatz einer Grafik als Aufzählungselement möglich. Letztere kann durch das Property `list-style-image` auch dem Listencontainer selbst zugewiesen werden und wird an die enthaltenen Listenelemente weitervererbt.

Verschachtelung von Listen

Das den Listenelementen vorangestellte Aufzählungssymbol zeigt die Hierarchiestufe im Falle der Verschachtelung mehrerer Listen an, wobei Listen auf der zweiten Ebene das Kreissymbol, solche auf der dritten Ebene das Quadratsymbol erhalten. Anschließend wird wieder das Scheibensymbol vergeben und so fort.

Ein Listencontainer `<ul>` darf laut DTD-Definition ausschließlich eine Folge von `<li>`-Containern enthalten, kann also *nicht* unmittelbar verschachtelt werden[64].

Folgendes Beispiel ist daher, obwohl verbreitet so gehandhabt und von Browsern akzeptiert, *falsch*:

```
<!-- falsche Verschachtelung von Listen: -->
<ul>
   <li>Stufe 1</li>
   <ul>
      <li>Stufe 2</li>
   </ul>
</ul>
```

Korrekt ist es vielmehr, die untergeordnete Liste innerhalb des letzten hierarchisch übergeordneten Listenelements einzufügen (der `<li>`-Container darf Blockelemente und somit auch Listencontainer enthalten):

```
<!-- richtige Verschachtelung von Listen: -->
<ul>
   <li>Stufe 1
   <ul>
      <li>Stufe 2</li>
   </ul>
   </li>
</ul>
```

Barrierefreiheit: Besondere Bedeutung besitzen ungeordnete Listen im zeitgenössischen Webdesign in Zusammenhang mit barrierearmen Websites, wo sie als Grundlage beliebig mit CSS präsentierbarer Navigationsmenüs dienen.

Beispiel: **[zB]**

Zwei einfache Listen mit voreingestellten Aufzählungssymbolen:

```
<ul type="disc">
   <li>Tomaten</li>
   <li>Basilikum</li>
</ul>
<ul type="square">
   <li>Rosen</li>
   <li>Tulpen</li>
   <li>Nelken</li>
</ul>
```

64 Die Definition lautet in der DTD zu HTML `<!ELEMENT UL - - (LI)+ -- unordered list -->` und in der XHTML-DTD analog `<!ELEMENT ul (li)+>`.

Anmerkung: Die Verwendung des `type`-Attributs ist »deprecated«. Das gewünschte Ergebnis kann ebenso gut mittels CSS erreicht werden. Hier wurde das HTML-Attribut nur aus Gründen der Einfachheit eingesetzt.

Verschachtelte Listen mit zwei Ebenen:

```
<ul>
   <li>Hammer</li>
   <li>Amboss
   <ul>
      <li>Damaststahl</li>
      <li>Sandwichblech</li>
      <li>Meteor-Eisen</li>
   </ul>
   </li>
</ul>
```

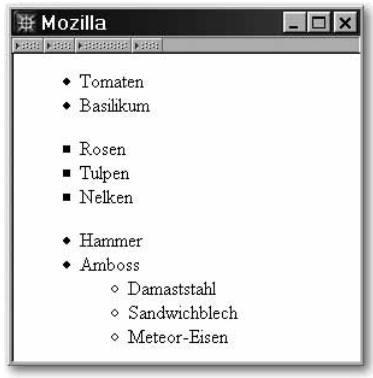

Abbildung 14.95 Ungeordnete Listen mit verschiedenen Bullets und Verschachtelung

14.91 var

Logische Auszeichnung: Das Element `<var>` (*variable*) bezeichnet eine Variable im Sinne einer mathematischen Größe oder einer Programmvariable.

Einführung	HTML 1.0
Elementtyp	Inlinecontainer
XHTML-Modul	Text
Kontext	Überall in Block- und Inlineelementen
Erlaubte Inhalte	Inlineelemente und Text
Offizielle W3C Spezifikationen	HTML 1.0, 2.0, 3.2, 4.0, 4.01; XHTML 1.0, 1.1, 2.0; XHTML Basic

Mobil-Spezifikationen	HTML 4.0 mobile, XHTML mobile profile *nicht enthalten in:* iMode XHTML 1.0, 1.1

Syntaxbeispiel:

```
<var> ... </var>
```

Defaultrendering:
CSS-Umsetzung des HTML-Defaults:

```
var        { display: inline; font-style: italic; }
```

14.91.1 Attribute (Übersicht)

Standardattribute

HTML	`class`, `dir`, `id`, `lang`, `style`, `title`
XHTML	`class`, `dir`, `id`, `xml:lang`, `style`, `title`

Eventhandler-Attribute

Common Events Collection	`onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup`
Intrinsic Events Collection	–
Internet Explorer Events	`onbeforecut`, `onbeforepaste`, `onblur`, `oncontextmenu`, `oncut`, `ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`, `onfocus`, `onhelp`, `onlosecapture`, `onpaste`, `onpropertychange`, `onreadystatechange`, `onresize`, `onselectstart`

Elementspezifische Attribute

W3C	–
Internet Explorer	Access-, Edit-Attribute

14.91.2 Proprietäre Attribute – Internet Explorer

Proprietäre Attributsammlungen – Internet Explorer

IE-Access-Attribute

`accesskey = character` (IE 5)

`hidefocus = "true" | "false"` (ab IE 5.5)

`tabindex = number` (IE 5)

IE-Edit-Attribute

`atomicselection = "true" | "false"` (IE 5)

`contenteditable = "inherit"` (Default) `| "false" | "true"`

`unselectable = "on" | "off"` (IE 5)

14.91.3 Erläuterung zum Element

Das Element `<var>` gehört zum XHTML-Textmodul und stellt eine logische Auszeichnung dar, die den Inhalt als Variable kennzeichnet. Die gewöhnliche Darstellung im Browser ist kursiv.

Das Element kann als zusätzliche Hervorhebung von Variablen innerhalb von `<code>`- oder `<pre>`-Containern eingesetzt werden, wenn diese zur Darstellung von Quellcode- oder Computerprogrammen dienen.

[zB] *Beispiel:*

```
<p>Als Variablenname wird gerne <var>x</var> verwendet.</p>
<p><var>A</var> + <var>B</var> = <var>C</var></p>
```

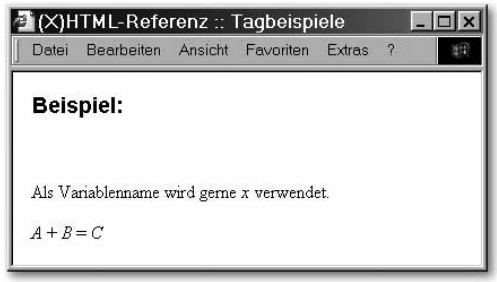

Abbildung 14.96 Die Darstellung von Variablen ist kursiv.

14.92 xmp

Historisch: Das Element `<XMP>` diente zur Kennzeichnung eines längeren »Beispiels« (im Allgemeinen eines Programmquelltexts).

14.92.1 Elementeigenschaften

Einführung	HTML Level 0
Elementtyp	Blockcontainer

XHTML-Modul	– (abgeschafft in HTML 4.0)
Kontext	Im `<body>`-Container
Erlaubte Inhalte	Text
Offizielle W3C Spezifikationen	HTML Level 0, HTML 1.0 HTML 2.0 (deprecated), HTML 3.2 (deprecated) **abgeschafft ab** HTML 4.0
Mobil-Spezifikationen	– (in keiner Mobil-Spezifikation enthalten)

Syntaxbeispiel:

```
<XMP> ... </XMP>
```

Defaultrendering:

CSS-Umsetzung des HTML-Defaults:

```
XMP  {display:block; }
```

14.92.2 Attribute (Übersicht)

Standardattribute

HTML	–
XHTML	(Element in XHTML nicht enthalten)

Eventhandler-Attribute

Common Events Collection	–
Intrinsic Events Collection	–
Internet Explorer Events	–

Elementspezifische Attribute

W3C	–

14.92.3 Erläuterung zum Element

Seit HTML 4.0 abgeschafft: Das Blockelement `<XMP>` (*example)* wurde eingeführt, um Codebeispiele in einem Webdokument einfügen zu können. Abgezielt wurde seinerzeit auf Beispiele von HTML-Quelltext innerhalb von Projektdokumentati-

onen, weshalb innerhalb des Elementcontainers als »Feature« kein HTML geparst wird. Eine Weißraumnormalisierung findet ebenfalls nicht statt: Innerhalb des Elementinhalts auftretende Tabulatoren, Leerzeichen Zeilenumbrüche und Leerzeilen werden unverändert im Browserfenster ausgegeben, Textinhalte in Monospace-Font dargestellt.

Historisches Element

Das Element wurde bereits in HTML 2.0 als »deprecated« eingestuft, jedoch erst mit HTML 4.0 abgeschafft. In XHTML existiert es nicht. Es wird in aktuellen Browsern jedoch noch unterstützt.

Als Ersatz für `<XMP>` kann das Element `<samp>` dienen, das ebenfalls ein Beispiel (*sample)* markiert.

Eine Auszeichnung als Quellcode kann alternativ auch mit dem Element `<code>` vorgenommen werden. Beide Elemente stellen ihre Inhalte ebenfalls in Monospace-Font dar, deaktivieren für ihren Inhaltsbereich jedoch den Parsing-Vorgang nicht.

Um Textbereiche im Quelltext ohne Weißraumnormalisierung auszugeben, kann der Container `<pre>` eingesetzt werden. Auch in diesem Fall erfolgt die Textausgabe in Monospace-Font.

[zB] *Beispiel:*

```
<p>Eine Überschrift wird in HTML folgendermaßen geschrieben:
<XMP><h1>Dies ist eine Überschrift Größe 1</h1></XMP></p>

<p>Der Umlaut "ä" wird in HTML folgendermaßen geschrieben:
<XMP>Dies ist der Umlaut &auml;</XMP></p>
```

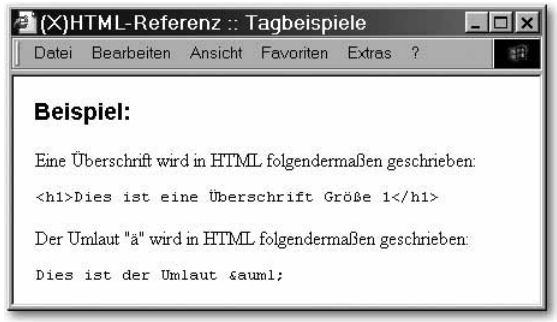

Abbildung 14.97 Innerhalb des <XMP>-Containers wird kein HTML ausgeführt.

Eine Referenz aller allgemeingültigen HTML-Attribute und ihrer Werte. Behandelt werden die für die Mehrzahl der Elemente gültigen spezifizierten Standard- und Eventhandler-Attribute sowie entsprechender Microsoft-Erweiterungen.

15 (X)HTML-Attributreferenz

In diesem Kapitel werden all jene Attribute beschrieben, die aus dem vorangegangenen Kapitel 14 aufgrund ihrer Allgemeingültigkeit für alle Elemente ausgegliedert wurden. Dies umfasst die so genannten Standard- oder Universalattribute wie auch die Eventhandler-Attribute, mit denen scriptgesteuerte Aktionen an Elemente gebunden werden können.

15.1 Standardattribute in HTML und XHTML

Seit HTML 4.0 existieren eine Reihe von **Standardattributen** (auch »Universalattribute«), die der Mehrzahl der HTML-Elemente zur Verfügung stehen. Diese Standardattribute ermöglichen es, einzelne Elementcontainer zu identifizieren, ihnen Styleklassen zuzuweisen und für sie individuell Sprache und Textrichtung festzulegen.

Allgemeine Attributsammlung – Die »Common Collection«

Die Standardattribute `id`, `class` und `title` werden auch als **Core-Attribute** bezeichnet und zählen entsprechend zur Core-Attributcollection der XHTML-Modularisierung. Die beiden Attribute `xml:lang` und `dir` sind Teil der **I18N-Attributsammlung** für Internationalisierung. `xml:lang` löst in XHTML 1.1 das in HTML 4.0 und 4.01 zu diesem gleichbedeutende Attribut `lang` endgültig ab – es unterscheidet sich von diesem lediglich dadurch, dass es im XML-Namensraum definiert ist. Das `style`-Attribut ist das einzige Mitglied der **Style-Attributsammlung**.

Zusammen mit den im folgenden Abschnitt beschriebenen **Eventhandler-Attributen** bilden die hier aufgeführten Attribute die so genannte **Common Collection** (»allgemeine Attributsammlung«), die fast allen Elementen aus HTML und XHTML zugeordnet ist. Zusätzlich können die Elemente individuelle, element-

spezifische Attribute besitzen. Diese werden getrennt in der Elementreferenz zusammen mit den Elementen behandelt.

Ausnahmen:

- Für `<br />`, `<applet>`, `<bdo>`, `<map>`, `<iframe>`, `<frame />` und `<frameset>` gilt nur die Core-Attributsammlung aus `id`, `class` und `title`.
- Ebenfalls eingeschränkt sind `<head>`- und `<title>`-Container, der `<style>`-Container und `<meta/>`, die nur die I18N-Sammlung erhalten. `<style>` besitzt zusätzlich das `title`-Attribut.
- `<param>` und `<basefont>` erhalten nur das `id`-Attribut, `<script>` und `<base/>` keines der Standardattribute.

class

Syntax
class = "classname [classname2 classname3 ... classnameN]"
XHTML-Modul
Core-Attributsammlung
Beschreibung
Das Attribut ist *case-sensitive*.
Das Attribut `class` dient zum Binden einer Stylesheetklasse an ein Element und erhält als Wert den Bezeichner einer solchen Klasse bzw., falls mehrere Klassen gleichzeitig auf das Element angewendet werden sollen, eine durch Leerzeichen getrennte Liste mehrerer Klassenbezeichner.
Nicht definiert für
`<html>`, `<head>`, `<title>`, `<base />`, `<basefont />`, `<meta />`, `<script>`, `<style>`, `<param />`

dir

Syntax
dir = "rtl" | "ltr" (Default)
XHTML-Modul
I18N-Attributsammlung
Beschreibung
Das Attribut ist nicht *case-sensitive*.
Das `dir`-Attribut legt die Basistextrichtung des Elementinhalts fest (links nach rechts, rechts nach links). Bei Blocklevel-Elementen werden die Textausrichtung zum Elternelement und das Setzen der Interpunktion beeinflusst, bei Tabellen die Rendering-Richtung der Tabellenzellen. Für das `<bdo>`-Element (s. dort) ist das `dir`-Attribut obligatorisch.

Anmerkung

Im Gegensatz zum sonst gängigen Verfahren vererbt sich die deklarierte Basistextrichtung eines Elements nicht an dessen Kindelemente.

Nicht definiert für

`<frameset>`, `<frame />`, `<iframe>`, `<applet>`, `<base />`, `<basefont />`, `<br />`, `<param />`, `<script>`

id

Syntax

id = "ID"

XHTML-Modul

Core-Attributsammlung

Beschreibung

Das Attribut ist *case-sensitive*.

Das `id`-Attribut erhält einen Wert vom Typ `ID`, der dokumentweit als individueller Identifier eines Elements fungiert, weshalb pro Dokument jeder `id`-Wert nur genau einmal vergeben werden darf. Das `id`-Attribut definiert, analog zum `name`-Attribut, gleichzeitig einen Fragment-Identifier, also das potentielle Ziel eines Sprungbefehls und kann gleichzeitig zum Binden von Stylesheetregeln an ein Element verwendet werden (per ID-Selektor `#idwert`).

Das universelle `id`-Attribut ersetzt in XHTML außer in wenigen Ausnahmen das vergleichbar eingesetzte `name`-Attribut. (Diese Ausnahmen sind Formularelemente, bei denen `name` für die Datenübertragung benötigt wird und deshalb weiterhin gestattet bleibt.)

Nicht definiert für

`<html>`, `<head>`, `<title>`, `<base />`, `<meta />`, `<script>`, `<style>`

lang

Syntax

lang = "sprachbezeichner"

XHTML-Modul

– (in XHTML abgelöst duch `xml:lang`)

Beschreibung

Das Attribut ist nicht *case-sensitive*.

Das `lang`-Attribut beschreibt in **HTML-Dokumenten** mittels einer Abkürzung nach RFC 1766 die Sprache[1] des Elementinhalts. Es gilt für Elemente, die Textinhalt (oder alternative Beschreibungen) enthalten können. Ist für ein Element (und auch für keines seiner Elternelemente) ein `lang`-Attribut gesetzt, so gilt die Sprache als »unknown« (Defaultwert). Die Sprachbeschreibung vererbt sich an Kindelemente, kann dort jedoch individuell durch `lang`-Attribute mit anderem Wert überschrieben werden.

1 Das `lang`-Attribut dient nicht zum Bestimmen der in einem `<script>`-Container geltenden Programmiersprache (es ist dort auch nicht definiert) – hierfür wird das `type`-Attribut verwendet.

Anmerkung

In XHTML-Dokumenten empfiehlt sich der parallele Einsatz von `xml:lang`. Im Konfliktfall hat `xml:lang` Vorrang vor `lang`. Wird das Dokument als `"text/html"` übertragen, kann wahlweise `lang` oder `xml:lang` eingesetzt werden; `xml:lang` verwendet man, wenn das Dokument als XML ausgeliefert wird. Da dies für XHTML 1.1 vorgeschrieben ist, *muss* hier `xml:lang` gewählt werden. (In XHTML 1.1 *entfällt* das `lang`-Attribut daher.)

Nicht definiert für:

`<frameset>`, `<frame />`, `<iframe>`, `<base />`, `<basefont />`, `<applet>`, `<br />`, `<script>`, `<param />`

style

Syntax

style = "CSS-Rule; [CSS-Rule2; CSS-Rule3; ... CSS-RuleN;]"

XHTML-Modul

Style-Attributsammlung

Beschreibung

Das `style`-Attribut wendet Styleangaben der für das Dokument vorgegebenen Stylesprache (in der Regel also CSS) unmittelbar auf ein Element an (Inlinestyle). Im Sinne einer strengen Trennung von Präsentation und Struktur sollte man derartige Styleangaben jedoch möglichst vermeiden.

Anmerkung

Falls erforderlich, kann man die Default-Stylesheetsprache dokumentweit mittels eines Meta-Tags setzen.

```
<meta http-equiv="Content-Style-Type" content="text/css"/>
```

Betroffen wären allerdings nur Inlinestyles, da sowohl für den Style-Container als auch für per Link-Tag eingebundene Stylesheets eine Typangabe mittels `type`-Attribut obligatorisch ist.

Nicht definiert für

`<html>`, `<head>`, `<title>`, `<base />`, `<basefont />`, `<meta />`, `<script>`, `<style>`, `<param />`

title

Syntax

title = "string"

XHTML-Modul

Core-Attributsammlung

Beschreibung

Das Attribut ist *case-sensitive*.

Das `title`-Attribut wird allgemein einem Element als Betitelung im Sinne einer Erläuterung beigegeben. In vielen Fällen wird der Attributwert bei Mouseover in Form eines **Tooltips** angezeigt, beispielsweise bei Links, Bildern, Akronymen oder Abkürzungen. Auch von Screenreadern kann das `title`-Attribut entsprechend genutzt werden (bei Akronymen wird es beispielsweise *anstelle* des abgekürzten Elementinhalts verlesen).

In Zusammenhang mit Verweisen auf alternative externe CSS-Stylesheets dient das `title`-Attribut des `<link>`-Elements als Beschreibung des jeweiligen Styles, wie er im Browsermenü gezeigt wird (z. B. in Mozilla unter View | Use style). Bei anderen Linkrelationen kann das `title`-Attribut außerdem von Suchmaschinen ausgewertet werden.

Nicht definiert für

`<html>`, `<head>`, `<title>`, `<base />`, `<basefont />`, `<meta />`, `<script>`, `<param />`

xml:lang

Syntax

xml:lang = "sprachbezeichner"

XHTML-Modul

I18N-Attributsammlung

Beschreibung

Das Attribut ist nicht *case-sensitive*.

Das `xml:lang`-Attribut beschreibt in **XHTML-Dokumenten** mittels einer Abkürzung nach RFC 1766 die Sprache des Elementinhalts. Es gilt für Elemente, die Textinhalt (oder alternative Beschreibungen) enthalten können. Ist für ein Element (und auch für keines seiner Elternelemente) ein `xml:lang`-Attribut gesetzt, so gilt die Sprache als »unknown« (Defaultwert). Die Sprachbeschreibung vererbt sich an Kindelemente, kann dort jedoch individuell durch `xml:lang`-Attribute mit anderem Wert überschrieben werden.

Anmerkung

Das Attribut befindet sich im XML-Namensraum, der aber als »vordefiniert« gilt und daher im XHTML-Dokument nicht extra vereinbart werden muss.

Nicht definiert für

`<frameset>`, `<frame />`, `<iframe>`, `<base />`, `<basefont />`, `<applet>`, `<br />`, `<param />`, `<script>`

15.2 Standard Eventhandler-Attribute

Eventhandler-Attribute dienen dazu, Interaktionen des Users mit der HTML-Seite zu erkennen und an ein bestimmtes HTML-Objekt gebundene Scriptprogramme (in der Regel JavaScript) auszuführen.

So erhält beispielsweise ein Link einen `onmouseover`-Eventhandler, um an diesem speziellen Objekt den Mouseover-Event aufzufangen:

```
<a href="#" onmouseover="alert('Mouseover-Event erkannt')">
Dieser Link reagiert auf Mouseover</a>
```

Als Wert des Eventhandler-Attributs sind Scriptprogrammbefehle (beispielsweise Funktionsaufrufe, aber auch beliebiger Programmquellcode) gestattet.

15.2.1 Eventhandler auf Applikations- und Interaktionsebene

Man unterscheidet anhand der Qualität des Triggervorgangs zwischen Eventhandlern auf **Applikationsebene** (Application Level Eventhandler) und solchen auf **Interaktionsebene** (Interaction Level Eventhandler).

Barrierefreiheit verlangt Eventhandler auf Applikationsebene

Unter dem Aspekt der Barrierefreiheit sind Trigger auf Applikationsebene vorzuziehen, da diejenigen auf Interaktionsebene nicht von allen Usern genutzt werden können (wenn sie beispielsweise Mausaktionen betreffen, der Nutzer jedoch auf Tastatureingaben beschränkt ist).

- **Applikationsebene**
  Zu dieser Gruppe von Eventhandlern zählt man die geräteunabhängigen Typen `onblur`, `onfocus` und `onselect`.

  Die mit diesen Handlern verknüpften Ereignisse sind nicht an bestimmte Useraktionen oder Eingabegeräte gebunden bzw. können durch verschiedene Handlungen hervorgerufen werden. So kann ein Focus-Event an einem Objekt gleichwertig durch Anklicken (Mausaktion) wie durch Anwahl des Objekts per Tabulator oder Accesskey (Keyboardaktion) ausgelöst werden.
- **Interaktionsebene**
  Zu dieser Gruppe von Eventhandlern zählen die geräteabhängigen Typen `onclick`, `ondblclick`, `onkeydown`, `onkeyup`, `onkeypress`, `onmouseover`, `onmouseout`, `onmouseup`, `onmousedown`, `onmousemove`.

  Die mit diesen Handlern verknüpften Ergeignisse setzen die Nutzung eines bestimmten Eingabegeräts (Maus bzw. Tastatur) voraus. Sofern nicht auf einen Eventhandler auf Applikationsebene ausgewichen werden kann, kann es ratsam sein, redundant vorzugehen und (bei gleichbleibendem aufgerufenen Script) gleichzeitig korrespondierende Eventhandler für das jeweils andere Eingabegerät zur Verfügung zu stellen. Dies ist allerdings nicht in jedem Falle möglich – einen Ansatz bietet die folgende Tabelle:

Tastaturevent	Mausevent
`onkeydown`	`onmousedown`
`onkeyup`	`onmouseup`
`- (kein Äquivalent)`	`onmousemove`
`onkeypress`	`onclick`
`- (kein Äquivalent)`	`ondblclick`

Tabelle 15.1 Korrespondierende Maus- und Tastaturevents

15.2.2 Allgemein einsetzbare Eventhandler

Die folgenden, hier beschriebenen Eventhandler-Attribute sind in allen aktuellen Browsern einsetzbar. Sie stehen entweder in Zusammenhang mit der **Common Event Collection** all denjenigen Elementen zur Verfügung, denen die Common Attribute Collection zugeordnet sind (dies sind beinahe alle), oder sie sind als Teil der **Intrinsic Event Collection** bestimmten Elementen gezielt zugeordnet.

Kleinschreibung der Attributnamen in XHTML

Beachten Sie, dass für valides XHTML die Eventhandler-Attribute durchgängig in **Kleinbuchstaben** geschrieben werden müssen. Die früher übliche Binnengroßschreibung (z. B. `onMouseDown`) ist nicht mehr gestattet.

Gültigkeit:

Die Eventhandler der Common Event Collection sind **nicht spezifiziert** für folgende Elemente: `<applet>`, `<base />`, `<basefont />`, `<bdo>`, `<br />`, `<font>`, `<frame />`, `<frameset>`, `<head>`, `<html>`, `<iframe>`, `<isindex />`, `<meta />`, `<param />`, `<script>`, `<style>`, `<title>`.

onblur

Beschreibung

Intrinsic Event Collection
Der `onblur`-Eventhandler feuert, wenn das überwachte Objekt den Fokus verliert, indem dieser auf ein anderes Objekt verlagert wird (z. B. ein ausgewähltes Objekt durch Tab-Befehl verlassen wird, ein Click-Event außerhalb des Objekts auftritt oder ein zweites Browserfenster geöffnet wird).

Für Texteingabefelder *folgt* deren Blur-Event unmittelbar dem Change-Event, sofern dessen Voraussetzungen gegeben sind. Der Focus-Event am neuen Objekt tritt nach dem Blur-Event am verlassenen Objekt auf.

Anwendbar auf

`<a>`, `<area />`, `<button>`, `<label>`, `<input />`, `<select>`, `<textarea>`, `<frame />`

Siehe auch

`onfocus`, `onfocusout` (IE = »Microsoft Erweiterung für Internet Explorer«)

onchange

Beschreibung

Intrinsic Event Collection
Der `onchange`-Eventhandler feuert, wenn ein Formularelement den Fokus verliert *und* sein Inhalt durch den User verändert wurde (d. h. eine Texteingabe oder -löschung erfolgt ist oder eine Option eines Select-Objekts gewählt wurde). Der Event tritt erst ein, wenn der Nutzer den Fokus vom überwachten Objekt nimmt, also nicht während, sondern erst nach der Eingabe.

Sofern das Objekt dies unterstützt, tritt unmittelbar *nach* dem Change-Event auch ein Blur-Event auf.

Anmerkung

Der Eventhandler wird vom Internet Explorer nicht für Passworteingabefelder `type="password"` unterstützt. Bei der Prüfung von Selectboxen muss der Eventhandler im `<select>`-Tag definiert sein, nicht in den Optionen.

Anwendbar auf

`<input type="text" />`, `<input type="password" />`, `<select>`, `<textarea>`

Siehe auch

`onsubmit`, `onreset`, `onselect`

onclick

Beschreibung

Common Event Collection
Der `onclick`-Eventhandler feuert, wenn der Nutzer das überwachte Element mit der linken Maustaste anklickt.

Dies entspricht in dieser Reihenfolge der Aufeinanderfolge von Mousedown- und Mouseup-Event über dem Objekt – tritt also z. B. der Mousedown-Event über dem Objekt auf, nicht aber der Mouseup-Event (weil der User die Maus bewegt hat), so erfolgt kein Click-Event.

Ist das Objekt zum Zeitpunkt des Klicks nicht im Fokus, so tritt unmittelbar *vor* dem Click-Event ein Focus-Event auf (sofern das Objekt diesen unterstützt).

Ist das angeklickte Objekt ein `<label>`-Element, so tritt *zuerst* ein Click-Event am Label-Objekt auf und danach ein *weiterer* Click-Event am Input-Objekt, auf das sich das `for`-Attribut des `<label>`-Tags bezieht (man sollte den Eventhandler daher nur an einem von beiden Objekten einsetzen).

Anmerkung

Ein Click-Event tritt ebenfalls ein durch Aktivieren eines Objekts mittels Accesskey oder Betätigen der ↵-Taste nach Eingabe in ein Formularfeld bzw. Auswahl einer Option einer Combo-Box. Unter dem Aspekt der Barrierefreiheit sollte man daher den Einsatz des `onclick`-Eventhandlers auf die Elemente beschränken, die von allen Nutzern auch über Tastaturbefehle zugänglich gemacht werden können (namentlich `<a>`, `<area />`, `<button>` und `<input />`).

Keyboardäquivalent: `onkeypress`

Anwendbar auf

Allgemein

Siehe auch

`ondblclick`

ondblclick

Beschreibung

Common Event Collection
Der `ondblclick`-Eventhandler feuert, wenn der Nutzer das überwachte Element mit der linken Maustaste doppelt anklickt (in Netscape auch mit der rechten Maustaste).

Unmittelbar *vor* einem DoubleClick-Event tritt auch ein Click-Event auf. Trotz dieser prinzipiellen Unterscheidbarkeit sollten diese Eventhandler besser nicht gemeinsam an einem Objekt eingesetzt werden.

Anmerkung

Für diesen Event existiert kein Keyboardäquivalent. Der `ondblclick`-Eventhandler kann nur über eine *Mausaktion* getriggert werden, der dem DoubleClick-Event vorangehende Click-Event gegebenenfalls auch über Auswahl des Objekts per Accesskey oder ⇆-Taste (der DoubleClick-Event tritt in diesem Falle nicht ein). Im Zweifelsfall ist der Barrierefreiheit halber der `onclick`-Eventhandler vorzuziehen.

Anwendbar auf

Allgemein

Siehe auch

`onclick`

onfocus

Beschreibung

Intrinsic Event Collection
Der `onfocus`-Eventhandler feuert, wenn das überwachte Objekt den Fokus bekommt (z. B. in ein Formulareingabeelement geklickt oder ein Link durch ⇆-Befehl ausgewählt wird).

Wird das Objekt durch Anklicken in den Fokus genommen, so tritt unmittelbar *nach* dem Focus-Event auch ein Click-Event auf (sofern das Objekt diesen unterstützt). War zuvor ein anderes Objekt im Fokus, so tritt an diesem *vor* dem Focus-Event am fokussierten Objekt ein Blur-Event auf.

Anwendbar auf

`<a>`, `<area />`, `<button>`, `<label>`, `<input />`, `<select>`, `<textarea>`, `<frame />`

Siehe auch

`onblur`, `onfocusout` (IE)

onkeydown

Beschreibung

Common Event Collection
Der `onkeydown`-Eventhandler feuert, wenn das überwachte Objekt im Fokus ist und der User eine Keyboardtaste drückt. Er kann zur Abfrage (oder Unterdrückung) einer Tastatureingabe eingesetzt werden.

Anmerkung

Feuert im Internet Explorer kontinuierlich, solange die Taste gedrückt gehalten wird.

Anwendbar auf
Allgemein
Siehe auch
`onkeyup`, `onkeypress`, `onhelp` (IE)

onkeypress

Beschreibung
Common Event Collection Der `onkeypress`-Eventhandler feuert, wenn das überwachte Objekt im Fokus ist und der User eine Keyboardtaste betätigt (drückt und wieder loslässt).
Anwendbar auf
Allgemein
Siehe auch
`onkeydown`, `onkeyup`, `onhelp` (IE)

onkeyup

Beschreibung
Common Event Collection Der `onkeyup`-Eventhandler feuert, wenn das überwachte Objekt im Fokus ist und der User eine gedrückte Keyboardtaste loslässt.
Anwendbar auf
Allgemein
Siehe auch
`onkeydown`, `onkeypress`, `onhelp` (IE)

onload

Beschreibung
Intrinsic Event Collection Der `onload`-Eventhandler feuert unmittelbar nach vollendetem Ladevorgang eines Dokuments. Er wird dafür an den `<body>`-Tag gebunden, im Falle eines Frameset-Dokuments an den `<frameset>`-Tag. Ebenfalls treten Onload-Events bei HTML-Elementen auf, die externe Dateien einbinden, wie z. B. dem `<applet>`-, `<embed>`-, `<img>`- oder `<object>`-Tag – hier muss darauf geachtet werden, dass ein dabei angesprochenes Script im Quelltext vor dem Objekt steht, da der Event unmittelbar beim Parsen eintritt, noch bevor die folgenden Objekte eingelesen werden.
Anmerkung
Der Eventhandler steht im Rahmen eines Framesets nicht für einzelne `<frame>`-Elemente zur Verfügung. Er muss für diesen Zweck an den `<body>` des im betreffenden Frame geladenen Dokuments gebunden werden.

Anwendbar auf
`<body>`, `<frameset>`, `<img />`
Siehe auch
`onunload`, `onbeforeunload` (IE)

onmousedown

Beschreibung
Common Event Collection Der `onmousedown`-Eventhandler wird getriggert, wenn der Nutzer die (linke) Maustaste über dem überwachten Objekt drückt. Keyboardäquivalent: `onkeydown`
Anwendbar auf
Allgemein
Siehe auch
`onmouseover`, `onmouseout`, `onmouseup`, `onmousemove`

onmousemove

Beschreibung
Common Event Collection Der `onmousemove`-Eventhandler wird getriggert, wenn der Nutzer den Mauszeiger innerhalb des Bereichs der Grenzen (Inhalt inkl. CSS-Padding) des überwachten Objekts bewegt.
Anmerkung
Für diesen Event existiert kein Keyboardäquivalent. Der `onmousemove` -Eventhandler kann nur über eine *Mausaktion* getriggert werden.
Anwendbar auf
Allgemein
Siehe auch
`onmouseover`, `onmouseout`, `onmousedown`, `onmouseup`

onmouseout

Beschreibung
Common Event Collection Der `onmouseout`-Eventhandler wird getriggert, wenn der Nutzer den Mauszeiger aus dem Bereich der Grenzen (Inhalt inkl. CSS-Padding) des überwachten Objekts hinausbewegt. Keyboardäquivalent: `onblur` (der User bewegt den Cursor bzw. Fokus per Tastatur vom Objekt weg)
Anwendbar auf
Allgemein

Siehe auch
`onmouseover`, `onmousedown`, `onmouseup`, `onmousemove`

onmouseover

Beschreibung
Common Event Collection Der `onmouseover`-Eventhandler feuert, wenn der Nutzer den Mauszeiger über das überwachte Objekt (Inhalt inkl. CSS-Padding) bewegt. Keyboardäquivalent: `onfocus` (der User bewegt den Cursor bzw. Fokus per Tastatur zum Objekt)
Anwendbar auf
Allgemein
Siehe auch
`onmouseout`, `onmousedown`, `onmouseup`, `onmousemove`

onmouseup

Beschreibung
Common Event Collection Der `onmouseup`-Eventhandler feuert, wenn der Nutzer die gedrückte (linke) Maustaste über dem überwachten Objekt loslässt. Keyboardäquivalent: `onkeyup`
Anwendbar auf
Allgemein
Siehe auch
`onmouseover`, `onmouseout`, `onmousedown`, `onmousemove`

onreset

Beschreibung
Intrinsic Event Collection Der `onreset`-Eventhandler feuert, wenn der Inhalt eines Formulars auf seine Ausgangswerte zurückgesetzt wird. Dies geschieht im Allgemeinen durch Betätigen des Reset-Buttons, kann aber auch durch ein Script über die `reset()`-Methode des Form-Objekts initiiert werden.
Anwendbar auf
`<form>`
Siehe auch
`onsubmit`, `onchange`, `onselect`

onselect

Beschreibung

Intrinsic Event Collection
Der `onselect`-Eventhandler feuert, wenn Text in einem Texteingabefeld oder Textinput eines Formulars ausgewählt wird oder der Umfang einer bestehenden Auswahl verändert wird.

Anmerkung

Der Eventhandler wird vom Internet Explorer auch für den `<body>`-Tag unterstützt, ist für diesen jedoch nicht offiziell spezifiziert.

Anwendbar auf

`<input type="..."/>` (außer `type="hidden"`), `<textarea>`

Siehe auch

`onsubmit`, `onreset`, `onchange`

onsubmit

Beschreibung

Intrinsic Event Collection
Der `onsubmit`-Eventhandler feuert, wenn die Daten eines Formulars verschickt werden. Dies geschieht im Allgemeinen durch Betätigen des Submit-Buttons, kann aber auch durch ein Script initiiert werden.

Anwendbar auf

`<form>`

Siehe auch

`onreset`, `onselect`, `onchange`

onunload

Beschreibung

Intrinsic Event Collection
Der `onunload`-Eventhandler feuert, wenn ein Dokument aus dem Browserfenster entladen (also ein neues Dokument an seine Stelle tritt) oder das Browserfenster geschlossen wird. Er wird dafür an den `<body>`-Tag gebunden, im Falle eines Frameset-Dokuments an den `<frameset>`-Tag.

Anmerkung

In Framesets tritt der Event dann ein, wenn der komplette Frameset entladen wird. Der Eventhandler überwacht keine einzelnen `<frame/>`-Elemente und kann nicht durch das Entladen des Dokuments eines Einzelframes getriggert werden. Er muss für diesen Zweck an den `<body>` des im betreffenden Frame befindlichen Dokuments gebunden sein.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onunload`, `onbeforeunload` (IE)

15.3 Microsoft-Erweiterungen – Eventhandler

Neben den offiziellen Eventhandler-Attributen werden von einigen Browserherstellern, an erster Stelle Microsoft, auch weitere Events propagiert und unterstützt, die deshalb an dieser Stelle kurz vorgestellt werden sollen.

[»] *Hinweis:* Die in Folge beschriebenen Eventhandler-Attribute sind nicht Teil der offiziellen HTML- oder XHTML-Spezifikationen.

Im Internet Explorer stehen daher auch für komplexere Mausaktionen wie Drag & Drop oder für Databinding-Events wie das Laden von Datenquellen entsprechende Handler zu Verfügung – diese sind als Teil von JScript[2] definiert.

Die meisten der nachfolgend beschriebenen Eventhandler sind im Wesentlichen auf alle HTML-Elemente im Dokument anwendbar.

Nicht eingesetzt werden können sie beim `<html>`-Element selbst und bei denjenigen Elementen, die vom Internet Explorer nicht unterstützt werden (z. B. `<abbr>`). Auch an Elemente, die zum Dokumentkopf gehören bzw. die im Dokument nicht sichtbar repräsentiert sind, können in der Regel keine Eventhandler erhalten (`<title>`, `<meta>`, `<script>`, `<style>`, aber auch `<input type="hidden">`, `<noscript>`, `<noframes>`).

onabort

Beschreibung
Microsoft **Bildladevorgang** Der `onabort`-Eventhandler feuert, wenn der User den Download einer Grafik abbricht – beispielsweise, indem der Stop-Button des Browsers betätigt wird oder das aktuelle Dokument noch während des Bildladevorgangs verlassen wird.
Anwendbar auf
`<img>`
Siehe auch
`onstop` **(IE)**

2 JScript: Die Microsoft-eigene Implementierung von JavaScript ist zwar kompatibel zu JavaScript (bzw. ECMA-Script), mit diesem aber nicht identisch. So geht JScript in einigen Punkten, nicht nur in Bezug auf die unterstützten Events, über JavaScript hinaus, ist aber in vollem Umfang nur im Microsoft-Umfeld (Internet Explorer, Windows Script Host, ASP, JScropt.NET) einsetzbar.

onactivate

Beschreibung

Microsoft **Objektaktivierung**
Der `onactivate`-Eventhandler feuert, wenn das überwachte, im Vorfeld nicht aktive Objekt zum aktiven Element wird. Dies geschieht entweder durch Anklicken, Anwahl per Tastatur (Tabulator, Acceskey) oder per Scriptbefehl (`setActive()`-Methode, JScript).

Sind *mehrere Dokumente* gleichzeitig geöffnet, so kann jedes von ihnen ein aktives Element beinhalten. Ein Dokument in bzw. aus dem Fokus zu nehmen, bedeutet nicht gleichzeitig die Änderung des Status dieses aktiven Elements.

Anmerkung

Auf den `<body>`-Container angewendet wird der `onactivate`-Handler *vor* dem `onload`-Handler getriggert. Ansonsten ist der Handler weitestgehend funktionsgleich mit dem `onfocus`-Handler.

Anwendbar auf

Allgemein

Siehe auch (alle IE)

`onbeforeactivate`, `onbeforedeactivate`, `ondeactivate`, `onfocusin`, `onfocusout`

onafterprint

Beschreibung

Microsoft **Print Events**
Der `onafterprint`-Eventhandler feuert unmittelbar, *nachdem* das mit ihm assoziierte Dokument entweder gedruckt oder seine Druckvorschau bereitgestellt wurde.

Anmerkung

Dieser Eventhandler wird in der Regel zusammen mit `onbeforeprint` eingesetzt, wobei in Verbindung mit dem Beforeprint-Event druckspezifische Änderungen am Dokument (z.B Expandieren von Seitenelementen) vorgenommen werden können, die beim Afterprint-Event wieder rückgängig gemacht werden.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onbeforeprint` (IE)

onafterupdate

Beschreibung

Microsoft **Databinding**
Der `onafterupdate`-Eventhandler feuert, wenn ein an das beobachtete Objekt gebundener Quelldatensatz (*data source object*) erfolgreich aktualisiert wurde.

Ein Afterupdate-Event setzt einen vorausgegangenen Beforeupdate-Event voraus, also den Eintritt einer Änderung am gebundenen Datensatz.

Anwendbar auf

<a>, <bdo>, <button>, , <frame />, <iframe>, <img />, <input type="checkbox" />, <input type="hidden" />, <input type="password" />, <input type="radio" />, <input type="text" />, <label>, <legend>, <marquee>, <rt>, <ruby>, <select>, , <textarea>

Siehe auch (alle IE)

onbeforeupdate, oncellchange, ondataavailable, ondatasetchanged, ondatasetcomplete, onerrorupdate, onrowsdelete, onrowenter, onrowexit, onrowsinserted

onbeforeactivate

Beschreibung

Microsoft **Objektaktivierung**
Der onbeforeactivate-Eventhandler feuert unmittelbar, bevor das überwachte Objekt zum aktiven Element wird, also vor dem onactivate-Eventhandler. Das Aktivieren eines Objekts geschieht entweder durch Anklicken, Anwahl per Tastatur (Tabulator, Acceskey) oder per Scriptbefehl (setActive()-Methode, JScript).

Der Beforeactivate-Event kann verwendet werden, um zu verhindern, dass ein Objekt aktiviert werden kann (ab Internet Explorer 6) – dies ist jedoch nur möglich, wenn die zu unterbrechende Aktivierung nicht per Script geschieht.

Anwendbar auf

Allgemein

Siehe auch (alle IE)

onctivate, onbeforedeactivate, ondeactivate, onfocusin, onfocusout

onbeforecopy

Beschreibung

Microsoft **Clipboard Events**
Der onbeforecopy-Eventhandler feuert am beobachteten Objekt, unmittelbar bevor eine getroffene Auswahl in die Zwischenablage kopiert wird.

Anwendbar auf

Alle Elemente, die im Browserfenster sichtbaren und auswählbaren Textinhalt besitzen dürfen, sowie den <img/>-Tag (nicht aber <applet> oder <object>).

Siehe auch (alle IE)

onbeforecut, onbeforepaste, oncopy, oncut, onpaste

onbeforecut

Beschreibung

Microsoft **Clipboard Events**
Der onbeforecut-Eventhandler feuert am beobachteten Objekt, unmittelbar bevor bei einer getroffenen Textauswahl bei Rechtsklick das Kontextmenü angezeigt wird.

Der Event wird benötigt, wenn das Ausschneiden von Teilen oder der Gesamtheit des Textinhalts eines Objekts über das Kontextmenü gestattet werden soll. Um den entsprechenden Kontextmenüpunkt zur Verfügung zu stellen, muss der Event den Wert `true` zurückgeben (dies wird über eine Funktion veranlasst).

Anwendbar auf

Alle Elemente, die im Browserfenster sichtbaren und auswählbaren Textinhalt besitzen dürfen, sowie den `<img/>`-Tag (nicht aber `<applet>` oder `<object>`).

Siehe auch (alle IE)

`oncut`, `oncopy`, `onbeforecopy`, `onpaste`, `onbeforepaste`

onbeforedeactivate

Beschreibung

Microsoft **Objektaktivierung**
Der `onbeforedeactivate`-Eventhandler feuert, unmittelbar bevor das aktive Element seinen Status verliert, indem ein anderes Objekt desselben Dokuments zum aktiven Dokument gemacht wird.

Der Eventhandler feuert dagegen *nicht*, wenn lediglich das aktuelle Dokument den Fokus verliert, da in diesem Fall der Zustand des aktiven Elements nicht geändert wird (jedes Dokument, unabhängig davon, ob im Fokus oder nicht, kann ein eigenes aktives Element beinhalten).

Anwendbar auf

Allgemein

Siehe auch (alle IE)

`onactivate`, `onbeforeactivate`, `ondeactivate`, `onfocusin`, `onfocusout`

onbeforeeditfocus

Beschreibung

Microsoft **Edit Mode**
Der `onbeforeeditfocus`-Eventhandler feuert, unmittelbar bevor das beobachtete Objekt für einen Editiervorgang ausgewählt wird. Der Edit Mode kann durch Setzen des MS-Attributs `contentEditable="true"` im `<body>` oder dem zu editierenden Dokumentabschnitt (z. B. einem Absatz), oder aber per Script aktiviert werden.

Anmerkung

Im **Browse Mode** (normaler Seitenmodus) ist der Eventhandler lediglich bei Formularinput-Elementen anwendbar. Der Beforeeditfocus-Event tritt beim Aktivieren des Inputs unmittelbar vor dem Focus-Event auf.

Anwendbar auf

Im Edit Mode allgemein, im Browse Mode nur `<input>`, `<textarea>`

Siehe auch

`onfocus`

onbeforepaste

Beschreibung

Microsoft **Clipboard Events**
Der `onbeforepaste`-Eventhandler feuert am beobachteten Objekt, unmittelbar bevor bei einer getroffenen Textauswahl bei Rechtsklick das Kontextmenü angezeigt wird.

Der Event wird analog zu `onbeforecut` benötigt, wenn das Einfügen in oder Ersetzen von Teilen oder der Gesamtheit des Textinhalts eines Objekts über das Kontextmenü gestattet werden soll. Um den entsprechenden Kontextmenüpunkt zur Verfügung zu stellen, muss der Event den Wert `true` zurückgeben.

Anwendbar auf

Alle Elemente, die im Browserfenster sichtbaren und auswählbaren Textinhalt besitzen dürfen, sowie den `<img/>`-Tag (nicht aber `<applet>` oder `<object>`).

Siehe auch (alle IE)

`oncut`, `oncopy`, `onbeforecopy`, `onbeforecut`, `onpaste`

onbeforeprint

Beschreibung

Microsoft **Print Events**
Der `onbeforeprint`- feuert, unmittelbar *bevor* das mit ihm assoziierte Dokument entweder gedruckt oder seine Druckvorschau bereitgestellt wurde.

Anmerkung

Dieser Eventhandler wird in der Regel zusammen mit `onafterprint` eingesetzt, wobei in Verbindung mit dem Beforeprint-Event druckspezifische Änderungen am Dokument (z.B Expandieren von Seitenelementen) vorgenommen werden können, die beim Afterprint-Event wieder rückgängig gemacht werden.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onafterprint` (IE)

onbeforeunload

Beschreibung

Microsoft **Dokument laden/entladen**
Der `onbeforeunload`-Eventhandler feuert, unmittelbar bevor ein Dokument oder ein kompletter Frameset aus dem Browserfenster entladen und durch ein neues Dokument ersetzt wird oder wenn das Browserfenster geschlossen werden soll. Der Beforeunload-Event tritt dabei unmittelbar vor dem Unload-Event auf.

Der Event kann eingesetzt werden, um dem User die Möglichkeit zu geben, das Entladen des aktuellen Dokuments bzw. das Schließen des Browsers abzubrechen, was mit dem `onunload`-Eventhandler nicht möglich ist.

Anmerkung

Der Eventhandler kann zwischen Verlassen und Neuladen des Dokuments nicht unterscheiden; ein durch den User veranlasstes Reload wird wie der Versuch, eine neue Seite zu laden bewertet, und triggert den Event.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onload`, `onunload`

onbeforeupdate

Beschreibung

Microsoft **Databinding**
Der `onbeforeupdate`-Eventhandler feuert am überwachten Objekt, wenn dessen Daten verändert wurden, bevor die mit ihm assoziierte Datenquelle aktualisiert wird.

Der Event tritt nach Veränderung der Daten ein, aber erst, wenn das betreffende Objekt den Fokus verliert oder das Dokument verlassen wird (dies ist analog zu `onchange` zu verstehen – die Datenänderung muss abgeschlossen sein).

Nach erfolgter Datenaktualisierung (falls diese nicht im Zuge des Beforeupdate-Events unterbunden wurde) tritt ein Afterupdate-Event ein.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`onafterupdate`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onrowsdelete`, `onrowenter`, `onrowexit`, `onrowsinserted`

onbounce

Beschreibung

Microsoft **Marquee**
Der `onbounce`-Eventhandler feuert an einem Marquee-Objekt, wenn dessen `behavior`-Attribut auf den Wert `"alternate"` gesetzt ist und der gescrollte Text einen Rand des Marquee-Bereiches erreicht hat.

```
<!-- Alertbox bei Erreichen des Marque-Randes: -->
<marquee onbounce="alert('Text hat Rand erreicht!')"
 behavior="alternate" loop="5">Lauftext</marquee>
```

Anmerkung

Die speziellen Marquee-Eventhandler werden in Opera und Mozilla/Firefox nicht unterstützt, obwohl diese das `<marquee>`-Element darstellen.

Anwendbar auf

`<marquee>`

Siehe auch (alle IE)

`onfinish`, `onstart`

oncellchange

Beschreibung

Microsoft **Databinding**
Der `oncellchange`-Event feuert, wenn sich der Inhalt der mit dem beobachteten Objekt verknüpften Datenquelle ändert.

Anwendbar auf

`<applet>`, `<bdo>`, `<object>`

Siehe auch (alle IE)

`onafterupdate`, `onbeforeupdate`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onrowsdelete`, `onrowenter`, `onrowexit`, `onrowsinserted`

oncontextmenu

Beschreibung

Microsoft **Mouse Event**
Der `oncontextmenu`-Eventhandler feuert, wenn der Nutzer mit der rechten Maustaste in das Dokument klickt, um das Kontextmenü zu öffnen. Gibt man im Eventhandler `false` zurück, kann das Kontextmenü deaktiviert werden.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onmousedown`

oncopy

Beschreibung

Microsoft **Clipboard Events**
Der `oncopy`-Eventhandler feuert, wenn der Nutzer am überwachten Objekt eine Textauswahl trifft und diese in die Zwischenablage kopiert.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`oncut`, `onpaste`, `onbeforecopy`, `onbeforepaste`, `onbeforecut`

oncut

Beschreibung

Microsoft **Clipboard Events**
Der `oncut`-Eventhandler feuert, wenn das beobachtete Objekt oder ein Teil seines Inhalts selektiert und ausgeschnitten (aus dem Dokument entfernt) wird. Hierfür muss das `contenteditable`-Attribut mit Wert `"true"` gesetzt sein.

Anwendbar auf

<body>, <frameset>

Siehe auch (alle IE)

oncopy, onbeforecopy, onpaste, onbeforepaste, onbeforecut

ondataavailable

Beschreibung

Microsoft **Databinding**
Der ondataavailable-Eventhandler feuert bei asynchroner Datenübertragung regelmäßig dann, wenn neue Daten vom Quellobjekt zur Verfügung gestellt werden.

Anwendbar auf

<body>, <frameset>

Siehe auch (alle IE)

onafterupdate, onbeforeupdate, oncellchange, ondatasetchanged, ondatasetcomplete, onerrorupdate, onrowsdelete, onrowenter, onrowexit, onrowsinserted

ondatasetchanged

Beschreibung

Microsoft **Databinding**
Der ondatasetchanged-Eventhandler feuert, wenn der Datensatz wechselt, den das Quellobjekt zur Verfügung stellt.

Anwendbar auf

<body>, <frameset>

Siehe auch (alle IE)

onafterupdate, onbeforeupdate, oncellchange, ondataavailable, ondatasetcomplete, onerrorupdate, onrowsdelete, onrowenter, onrowexit, onrowsinserted

ondatasetcomplete

Beschreibung

Microsoft **Databinding**
Der ondatasetcomplete-Eventhandler feuert, wenn der Datensatz am Quellobjekt vollständig zur Verfügung steht.

Anwendbar auf

<body>, <frameset>

Siehe auch (alle IE)

onafterupdate, onbeforeupdate, oncellchange, ondataavailable, ondatasetchanged, onerrorupdate, onrowsdelete, onrowenter, onrowexit, onrowsinserted

ondeactivate

Beschreibung

Microsoft **Objektaktivierung**
Der `ondeactivate`-Eventhandler feuert am beobachteten Objekt, wenn dieses aktiv ist und der aktive Status auf ein anderes Objekt übergeht.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onactivate`, `onbeforeactivate`, `onbeforedeactivate`, `onfocusin`, `onfocusout`

ondrag

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondrag`-Eventhandler feuert permanent am beobachteten Objekt, solange dieses per Mausaktion gezogen wird.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`

ondragend

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondragend`-Eventhandler feuert am beobachteten Objekt, wenn nach Ende des Drag-Vorgangs die Maus losgelassen wird.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondrag`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`

ondragenter

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondragenter`-Eventhandler feuert am beobachteten Objekt, wenn dieses mögliches Ziel eines Drop-Vorgangs ist und der User ein Objekt in dessen Bereich zieht.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondrag`, `ondragend`, `ondragleave`, `ondragover`, `ondragstart`, `ondrop`

ondragleave

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondragleave`-Eventhandler feuert am beobachteten Objekt, wenn dieses mögliches Ziel eines Drop-Vorgangs ist und der User ein Objekt aus dessen Bereich hinauszieht.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondrag`, `ondragend`, `ondragenter`, `ondragover`, `ondragstart`, `ondrop`

ondragover

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondragover`-Eventhandler feuert fortwährend am beobachteten Objekt, wenn dieses mögliches Ziel eines Drop-Vorgangs ist und solange der User ein Objekt innerhalb dessen Bereich zieht.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragstart`, `ondrop`

ondragstart

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondragstart`-Eventhandler feuert am beobachteten Objekt, wenn der User beginnt, das Objekt zu ziehen, oder mit gedrückter Maustaste eine Textauswahl innerhalb des Objektes vornimmt.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondrop`

ondrop

Beschreibung

Microsoft **Mouse Drag Events**
Der `ondrop`-Eventhandler feuert am beobachteten Objekt, wenn dieses erlaubtes Ziel eines Drop-Vorgangs ist und der User über dem Objekt ein gezogenes Objekt fallen lässt.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`ondrag`, `ondragend`, `ondragenter`, `ondragleave`, `ondragover`, `ondragstart`

onerror

Beschreibung

Microsoft **Ladevorgang (Objekt)**
Der `onerror`-Eventhandler feuert am beobachteten Objekt, wenn während des Ladevorgangs dieses Objekts ein Fehler auftritt.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch:

keine vergleichbaren Eventhandler

onerrorupdate

Beschreibung

Microsoft **Databinding**
Der `onerrorupdate`-Eventhandler feuert am beobachteten Objekt, wenn dieses an eine Datenquelle gebunden ist und beim Aktualisieren des gebundenen Datensatzes an der Datenquelle ein Fehler auftritt.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`onafterupdate`, `onbeforeupdate`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onrowsdelete`, `onrowenter`, `onrowexit`, `onrowsinserted`

onfilterchange

Beschreibung

Microsoft **Darstellungsfilter**
Der `onfilterchange`-Eventhandler feuert am beobachteten Objekt, wenn ein auf das Objekt angewendeter visueller Filter eine Zustandsänderung vollendet hat.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

keine vergleichbaren Eventhandler

onfinish

Beschreibung

Microsoft **Marquee**
Der `onfinish`-Eventhandler feuert an einem Marquee-Objekt, wenn dessen Schleifendurchlauf beendet ist.

```
<!-- Alertbox bei Ende des Marquee-Durchlaufs: -->
<marquee onfinish="alert('Marquee Durchlauf beendet!')"
 behavior="alternate" loop="5">Lauftext</marquee>
```

Anmerkung

Die speziellen Marquee-Eventhandler werden in Opera und Mozilla/Firefox nicht unterstützt, obwohl diese das `<marquee>`-Element darstellen.

Anwendbar auf
`<body>`, `<frameset>`
Siehe auch (alle IE)
`onbounce`, `onstart`

onfocusin

Beschreibung
Microsoft **Objektaktivierung** Der `onfocusin`-Eventhandler feuert am beobachteten Objekt unmittelbar, bevor diesem der Fokus übertragen wird (zuvor tritt am deaktivierten Objekt ein Focusout-Event auf).
Anwendbar auf
`<body>`, `<frameset>`
Siehe auch (alle IE bis auf *)
`onactivate`, `onbeforeactivate`, `onbeforedeactivate`, `ondeactivate`, `onfocus*`, `onfocusout`

onfocusout

Beschreibung
Microsoft **Objektaktivierung** Der `onfocusout`-Eventhandler feuert am beobachteten Objekt unmittelbar, bevor diesem der Fokus entzogen wird (unmittelbar danach tritt am aktivierten Objekt ein Focusin-Event auf).
Anwendbar auf
`<body>`, `<frameset>`
Siehe auch (alle IE bis auf *)
`onactivate`, `onbeforeactivate`, `onbeforedeactivate`, `ondeactivate`, `onfocus*`, `onfocusin`

onhelp

Beschreibung
Microsoft **Keyboard Events** Der `onhelp`-Eventhandler feuert am beobachteten Objekt wenn der Nutzer die F1-Taste betätigt. Das Browserfenster muss zu diesem Zeitpunkt das aktive Fenster sein.
Anwendbar auf
`<body>`, `<frameset>`
Siehe auch
`onkeyup`, `onkeydown`, `onkeypress`

onlayoutcomplete

Beschreibung

Microsoft **Druckvorgang**
Der `onlayoutcomplete`-Eventhandler feuert, wenn der Layoutvorgang eines Druck- oder Druckvorschauvorgangs das Layout des aktuellen Layoutrechtecks beendet hat (d.h. mit Inhalten aus dem Quelldokument gefüllt hat).

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

keine vergleichbaren Eventhandler

onlosecapture

Beschreibung

Microsoft **Mouse Events**
Der `onlosecapture`-Eventhandler feuert, wenn ein Objekt, das Mouse Events auffängt (»mouse capture«, z. B. mit der `setCapture`-Methode) diese wieder freigibt (beispielsweise mit der `releaseCapture`-Methode). Diese Script-Methoden sind auf den Internet Explorer beschränkt.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

(keine vergleichbaren Eventhandler)

onmouseenter

Beschreibung

Microsoft **Mouse Events**
Der `onmouseenter`-Eventhandler feuert, wenn der Nutzer den Mauszeiger von außerhalb in den Bereich des beobachteten Objekts bewegt.

Im Unterschied zum Mouseover-Event unterliegt der Mouseenter-Event nicht dem **Eventbubbling**[3], wird also *nicht* von Unterobjekten des beobachteten Objekts nach oben weitergereicht (was bei Mouseover der Fall ist – hier »bubbelt« ein am Unterobjekt eingetretenes Ereignis in der Hierarchie nach oben, bis es aufgefangen wird).

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onmouseover`

3 Es scheint tatsächlich keinen adäquaten deutschsprachigen Ausdruck hierfür zu geben.

onmouseleave

Beschreibung

Microsoft **Mouse Events**
Der `onmouseleave`-Eventhandler feuert, wenn der Nutzer den Mauszeiger aus den Bereich des beobachteten Objekts heraus nach außerhalb bewegt.

Analog zum Mouseenter-Event unterliegt der Mouseleave-Event nicht dem **Eventbubbling** und wird dementsprechend nicht von Unterobjekten nach oben weitergereicht. Hierin unterscheidet er sich vom Mouseout-Event.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch

`onmouseout`

onmousewheel

Beschreibung

Microsoft **Mouse Events**
Der `onmousewheel`-Eventhandler feuert, wenn der Nutzer das Scrollrad der Maus betätigt.

Anwendbar auf

Alle

Siehe auch

`onscroll` (IE)

onmove

Beschreibung

Microsoft **Animation**
Der `onmove`-Eventhandler feuert, wenn sich die Koordinaten des beobachteten Objekts ändern, dieses sich also bewegt (per Scripting oder Drag). Das Objekt muss absolut oder relativ positioniert sein. Das Ereignis tritt nur am bewegten Objekt selbst auf, nicht an einem seiner mit ihm mitbewegten Unterobjekte.

Anwendbar auf

Alle

Siehe auch (alle IE)

`onmoveend`, `onmovestart`

onmoveend

Beschreibung

Microsoft **Animation**
Der `onmove`-Eventhandler feuert, wenn die Bewegung des beobachteten Objekts beendet ist. Das Objekt muss absolut oder relativ positioniert sein. Das Ereignis tritt nur am bewegten Objekt selbst auf, nicht an einem seiner mit ihm mitbewegten Unterobjekte.

Anwendbar auf

Alle

Siehe auch (alle IE)

`onmovestart`, `onmove`

onmovestart

Beschreibung

Microsoft **Animation**
Der `onmovestart`-Eventhandler feuert, wenn das beobachtete Objekt sich zu bewegen beginnt (per Scripting oder Drag). Das Objekt muss absolut oder relativ positioniert sein. Das Ereignis tritt nur am bewegten Objekt selbst auf, nicht an einem seiner mit ihm mitbewegten Unterobjekte.

Anwendbar auf

Alle

Siehe auch (alle IE)

`onmoveend`, `onmove`

onpaste

Beschreibung

Microsoft **Clipboard Events**
Der `onpaste`-Eventhandler feuert, wenn der Inhalt der Zwischenablage in das beobachtete Objekt hineinkopiert wird.

Anwendbar auf

`<body>`, `<frameset>`

Siehe auch (alle IE)

`oncopy`, `onbeforecopy`, `onbeforepaste`, `onbeforecut`

onpropertychange

Beschreibung

Microsoft **Zustandsänderung**
Der `onpropertychange`-Eventhandler feuert, wenn eine Eigenschaft (Property) des beobachteten Objekts per Script verändert wird.

Anwendbar auf
Alle
Siehe auch
Keine vergleichbaren Eventhandler

onreadystatechange

Beschreibung
Microsoft **Zustandsänderung** Der `onreadystatechange`-Eventhandler feuert, wenn sich der Zustand des überwachten Objekts geändert hat. Dies tritt ein, wenn ein Objekt Daten lädt oder mit ihnen ein DHTML-Verhalten verknüpft wird. Der Eventhandler kann eingesetzt werden, um den Zustand des beobachteten Objekts zu prüfen, um nicht durch vorzeitiges Verwenden eines verknüpften Behaviors oder Data-Objects einen Scriptfehler zu verursachen.
Anmerkung
Der `onreadystatechange`-Eventhandler ist inzwischen auch in allen von Mozilla abgeleiteten Browsern sowie Opera implementiert und dient hier als Grundlage für Ajax-Anwendungen, um deren Serverantwort auszuwerten.
Anwendbar auf
Allgemein, aber auch: `<html>`, `<input type="hidden">`, `<noframes>`, `<script>`, `<noscript>`
Siehe auch
`onload`

onresize

Beschreibung
Microsoft **Edit Mode** Der `onresize`-Eventhandler feuert, wenn sich die Abmessungen des überwachten Objekts geändert haben (Breite, Höhe; auch entsprechende CSS-Angaben – Abmessungsangaben oder eine Positionierungsangabe müssen für das Objekt jedoch vorliegen).
Anwendbar auf
Alle positionierbaren Elemente
Siehe auch (alle IE)
`onresizeend`, `onresizestart`

onresizeend

Beschreibung
Microsoft **Edit Mode** Der `onresizeend`-Eventhandler feuert, wenn die Skalierung des überwachten Objekts abgeschlossen ist (Änderung von Breite, Höhe oder entsprechenden CSS-Angaben – Abmessungsangaben oder eine Positionierungsangabe müssen für das Objekt jedoch vorliegen).
Anwendbar auf
Alle positionierbaren Elemente
Siehe auch (alle IE)
`onresize`, `onresizestart`

onresizestart

Beschreibung
Microsoft **Edit Mode** Der `onresizestart`-Eventhandler feuert, wenn mit einem Skalierungsvorgang des überwachten Objekts begonnen wird (Änderung von Breite, Höhe oder entsprechenden CSS-Angaben – Abmessungsangaben oder eine Positionierungsangabe müssen für das Objekt jedoch vorliegen).
Anwendbar auf
Alle positionierbaren Elemente
Siehe auch (alle IE)
`onresize`, `onresizeend`

onrowenter

Beschreibung
Microsoft **Databinding** Der `onrowenter`-Eventhandler feuert, wenn an das beobachtete Objekt eine Datenquelle gebunden ist, nachdem von der Datenquelle ein neuer Datensatz (Row) zur Verfügung gestellt wurde.
Anwendbar auf
`<applet>`, `<object>`, `<xml>`
Siehe auch (alle IE)
`onafterupdate`, `onbeforeupdate`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onrowsdelete`, `onrowexit`, `onrowsinserted`

onrowexit

Beschreibung

Microsoft **Databinding**
Der `onrowexit`-Eventhandler feuert, wenn an das beobachtete Objekt eine Datenquelle gebunden ist, bevor sich der aktuell dargestellte Datensatz ändert, weil von der Datenquelle ein neuer Datensatz (Row) zur Verfügung gestellt wurde.

Anwendbar auf

`<applet>`, `<object>`, `<xml>`

Siehe auch (alle IE)

`onafterupdate`, `onbeforeupdate`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onrowsdelete`, `onrowenter`, `onrowsinserted`

onrowsdelete

Beschreibung

Microsoft **Databinding**
Der `onrowsdelete`-Eventhandler feuert, wenn an das beobachtete Objekt eine Datenquelle gebunden ist, bevor aus der Datenquelle ein oder mehrere Datensätze (Rows) gelöscht werden.

Anwendbar auf

`<applet>`, `<object>`, `<xml>`

Siehe auch (alle IE)

`onafterupdate`, `onbeforeupdate`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onrowenter`, `onrowexit`, `onrowsinserted`

onrowsinserted

Beschreibung

Microsoft **Databinding**
Der `onrowsinserted`-Eventhandler feuert, wenn an das beobachtete Objekt eine Datenquelle gebunden ist, nachdem in die Datenquelle ein oder mehrere Datensätze (Rows) eingefügt wurden.

Anwendbar auf

`<applet>`, `<object>`, `<xml>`

Siehe auch (alle IE)

`onafterupdate`, `onbeforeupdate`, `oncellchange`, `ondataavailable`, `ondatasetchanged`, `ondatasetcomplete`, `onerrorupdate`, `onrowsdelete`, `onrowenter`, `onrowexit`

onscroll

Beschreibung

Microsoft **Fenster Event**
Der `onscroll`-Eventhandler feuert bei einem Scrollvorgang am Browserfenster oder am beobachteten Objekt, falls dessen Inhalt scrollbar ist. Ein Scroll-Event kann durch Maus-drag- und -scroll- oder Keyboardaktionen (Pfeiltasten, Bildtasten), Klicken des Scrollpfeiles oder Scriptbefehle ausgelöst werden.

Anwendbar auf

`<applet>`, `<bdo>`, `<body>`, `<div>`, `<embed>`, `<map>`, `<marquee>`, `<object>`, `<table>`, `<textarea>`

Siehe auch

`onmousewheel` (IE)

onselectstart

Beschreibung

Microsoft **Objektaktivierung**
Der `onselectstart`-Eventhandler feuert am beobachteten Objekt, wenn der Nutzer beginnt, dessen Inhalte zu selektieren oder mit der Maus ein Auswahlrechteck zu ziehen.

Anwendbar auf

Alle Elemente mit auswählbarem Inhalt

Siehe auch

`onselect`

onstart

Beschreibung

Microsoft **Marquee**
Der `onstart`-Eventhandler feuert an einem Marquee-Objekt, wenn dessen Schleifendurchlauf begonnen hat.

```
<!-- Alertbox bei Beginn des Marquee-Durchlaufs: -->
<marquee onstart="alert('Marquee Durchlauf gestartet!')"
 behavior="alternate" loop="5">Lauftext</marquee>
```

Anmerkung

Die speziellen Marquee-Eventhandler werden in Opera und Mozilla/Firefox nicht unterstützt, obwohl diese das `<marquee>`-Element darstellen.

Anwendbar auf

`<marquee>`

Siehe auch (alle IE)

`onbounce`, `onfinish`

onstop

Beschreibung
Microsoft **Ladevorgang** Der `onstop`-Eventhandler feuert, wenn der User den Ladevorgang der Seite abbricht, indem er den Stop-Button des Browsers betätigt oder die Seite vor Beenden des Ladens verlässt.
Anwendbar auf
`<body>`
Siehe auch
`onabort` (IE)

15.4 Microsoft-Erweiterungen – Standardattribute

Microsoft erweitert die Anzahl (beinahe) allgemein zuweisbarer Attribute, um in seinen Browsern und dem hauseigenen MSHTML Editor spezielle Features zu unterstützen. Die Attribute lassen sich nach Aufgabenbereich wie folgt in drei Gruppen einteilen:

- **Accessibility**
  Der Internet Explorer unterstützt für die Mehrzahl der HTML-Elemente die Accessibility-Attribute `accesskey` und `tabindex`. Auch das Attribut `hidefocus` kann hier eingeordnet werden, dient jedoch nicht direkt der Accessibility.
- **Databinding**
  Die Databinding-Attribute wurden mit Internet Explorer 4.0 eingeführt und mit der Version 5.0 erweitert. Sie dienen dem Zuweisen von Datenquellen an Elemente, denen auf diese Weise externe Datenquellen zugewiesen werden können. Zu den Databinding-Attributen zählen `datasrc`, `datapagesize`, `datafld`, `dataformatas`.
- **Editing**
  Die drei Editing-Attribute `atomicselection`, `contenteditable` und `unselectable` können ebenfalls der Mehrzahl der HTML-Elemente zugewiesen werden. Ihre Wirkung beschränkt sich (abgesehen vom Attribut `contenteditable`) auf die Umgebung des MSHTML-Editors.

Alle anderen Browserfirmen ignorieren diese Attribute, da sie nicht im offiziellen HTML-Standard enthalten sind.

15.4.1 Access-Attribute

Die Access-Attribute erlauben das Fokussieren oder Selektieren eines Elements mit Keyboard-Shortcut oder Tabulator. Die Attribute `accesskey` und `tabindex`

sind auch im Rahmen der regulären HTML-Spezifikation definiert, allerdings für weniger Elemente.

Das Attribut `hidefocus` ist ebenfalls in diese Kategorie aufgenommen, da es die Präsentationswirkung von Objektfokussierung behandelt und ähnlich global einsetzbar ist, wie die beiden anderen Access-Attribute – streng genommen ist es für Accessibility-Zwecke jedoch bedeutungslos.

accesskey

Syntax
accesskey = "keyboard-character"
Plattform
Internet Explorer ab 5.0
Beschreibung
Das Attribut `accesskey` legt im Sinne der Accessibility ein (einzelnes) alphanumerisches Tastaturzeichen fest, das im Sinne eines Keyboard-Shortcuts den *Fokus* auf das Element legt – unter Windows bei gleichzeitig gedrückter Alt-Taste. Je nach Art des Elements hat dies verschiedene Folgewirkungen – ein Link wird ausgeführt, eine Checkbox oder ein Radiobutton umgeschaltet etc.
Anwendbar auf
Alle Elemente mit Ausnahme von `<base/>`, `<basefont/>`, `<form>`, `<frame/>`, `<frameset>`, `<html>`, `<head>`, `<iframe>`, `<link/>`, `<meta/>`, `<noscript>`, `<param>`, `<script>`, `<style>`, `<title>`

hidefocus

Syntax
hidefocus = "true" | "false"
Plattform
Internet Explorer ab 5.5
Beschreibung
Das `hidefocus`-Attribut bestimmt, ob die gestrichelte Focus-Outline um das Objekt im Falle des Fokus (z. B. per Tabulator) unterdrückt werden soll. Konsequenterweise ist das Attribut überall dort anwendbar, wo auch das `tabindex`-Attribut gesetzt werden kann (sowie bei einigen weiteren Elementen).
Anwendbar auf
Alle Elemente mit Ausnahme von `<base/>`, `<basefont/>`, `<html>`, `<head>`, `<link/>`, `<meta/>`, `<noscript>`, `<param>`, `<script>`, `<style>`, `<title>`

tabindex

Syntax

tabindex = "number"

Plattform

Internet Explorer ab 5.0

Beschreibung

Das `tabindex`-Attribut darf im Internet Explorer eine positive oder negative Ganzzahl im Bereich von –32767 bis (+) 32767 erhalten. Die Zahl legt in aufsteigender Richtung die Position des Elements in der Tabulatorwahl-Reihenfolge (⇆-Taste) fest.

Elemente mit **negativer** Tabindex-Zahl werden aus der Tabreihenfolge **entfernt**, auch wenn sie ansonsten enthalten wären. Hiervon betroffen sind die Elemente `<a>`, `<body>`, `<button>`, `<frame/>`, `<iframe>`, `<img/>`, `<input/>`, `<isindex/>`, `<object>`, `<select>` und `<textarea>`.

Anwendbar auf

Alle Elemente mit Ausnahme von `<base/>`, `<basefont/>`, `<embed>`, `<html>`, `<head>`, `<link/>`, `<meta/>`, `<noscript>`, `<param>`, `<script>`, `<style>`, `<title>`

15.4.2 Databinding-Attribute

Die Databinding-Attribute legen Datenquellen und das Verhältnis der Data-Consumer zu den Datenquellen fest.

datasrc

Syntax

datasrc = "datenquellen-id"

Plattform

Internet Explorer ab 4.0

Beschreibung

Das `datasrc`-Attribut gibt in einem **Data-Consumerobjekt** den ID-Bezeichner der zu adressierenden Datenquelle (»data source object«) an, von der das Objekt die Daten beziehen soll. Dem ID-Bezeichner wird vergleichbar mit einem Fragment-Identifier das Hash-Zeichen # vorangestellt. Data-Consumer können eine Vielzahl von HTML-Elementen sein; als Datenquell-Objekte wird in der Regel ein `<object>`-Element eingesetzt.

Als Property lese- und schreibbar. Kein Defaultwert.

Beispiel

```
<!-- Das Data Source Object: -->
<object id="kundendaten" classid=" ...(4)">
  <param name="DataURL" value="daten.txt">
  <param name="UseHeader" value="true">
```

4 Für tabellarische Daten lautet der erforderliche Wert: `classid= "333C7BC4-43460F-11D0-BD04-0080C7055A83"`

Beispiel

```
</object>

<!-- Der Data Consumer: -->
<table datasrc="#kundendaten"> ... </table>
```

Anwendbar auf

`<a>`, `<applet>`, `<button>`, `<div>`, `<frame/>`, `<iframe>`, `<img/>`, `<input type="button"/>`, `<input type="checkbox"/>`, `<input type="hidden"/>`, `<input type="image"/>`, `<input type="password"/>`, `<input type="radio"/>`, `<input type="text"/>`, `<label>`, `<marquee>`, `<select>`, `<span>`, `<table>`, `<textarea>`

datafld

Syntax

datafld = "feldname"

Plattform

Internet Explorer ab 4.0

Beschreibung

Das `datafld`-Attribut gibt den Namen des Datenfeldes eines Datensatzes der über das `datasrc`-Attribut genannten Datenquelle an, das im Rahmen des Databindings eingesetzt werden soll.

Als Property lese- und schreibbar. Kein Defaultwert.

Beispiel

```
<table datasrc="#kundendaten">
<tr><td>
  <span datafld="vorname"></span>
  <span datafld="nachname"></span>
</td></tr>
</table>
```

Anwendbar auf

`<a>`, `<applet>`, `<button>`, `<div>`, `<fieldset>`, `<frame/>`, `<iframe>`, `<img/>`, `<input type="button"/>`, `<input type="checkbox"/>`, `<input type="hidden"/>`, `<input type="image"/>`, `<input type="password"/>`, `<input type="radio"/>`, `<input type="text"/>`, `<label>`, `<marquee>`, `<select>`, `<span>`, `<textarea>`

dataformatas

Syntax

dataformatas = "text" | "localized-text" (ab IE 5.01) | "HTML"

Plattform

Internet Explorer ab 4.0

Beschreibung

Das `dataformatas`-Attribut (»data format as«) legt fest, wie die an ein Objekt gebundenen Daten interpretiert werden sollen – als reine Textdaten (Default) oder als HTML geparst. Die dritte Alternative (ab IE 5.01) besteht in der Angabe »localized text«, die Texte gemäß der Spracheinstellung des Clientrechners formatiert – dies betrifft im Wesentlichen Anführungszeichen.

Als Property lese- und schreibbar.

Beispiel

Wird eine Datenquelle an ein `<div>`-Element gebunden und soll in dessen Inhalt als HTML und nicht als ASCII-Text ausgegeben werden, so muss unbedingt `dataformatas="HTML"` angegeben werden:

```
<div datasrc="#datenquellen-id" dataformatas="html">...</div>
```

Anwendbar auf

`<div>`, `<span>`, `<marquee>`, `<label>`, `<legend>`, `<button>`, `<input type="button">`

datapagesize

Syntax

datapagesize = "number"

Plattform

Internet Explorer ab 4.0

Beschreibung

Das Attribut `datapagesize` legt die maximale Anzahl an Datensätzen fest, die in einer Tabelle mit Databinding zu gegebenem Zeitpunkt angezeigt werden sollen

Als Property lese- und schreibbar.

Beispiel

```
<!-- Maximal 10 Datensätze ausgeben: -->
<table datasrc="#kundendaten" datapagesize="10">
...
</table>
```

Anwendbar auf

`<table>`

15.4.3 Edit-Attribute

atomicselection

Syntax

atomicselection = "true" | "false" (Default)

Plattform

MSHTML Editor (Internet Explorer ab 5.5)

Beschreibung

Das Attribut `atomicselection` kann für alle Elemente eingesetzt werden, die über einen selektierbaren Textinhalt verfügen, oder aber (wie Grafiken) insgesamt selektierbar sind. Zweck des Attributs ist es, den jeweiligen Elementinhalt nur *vollständig* auswählbar zu machen. Im Rahmen eines Editiervorgangs soll damit vermieden werden, dass entsprechende Inhalte lediglich in Teilen geändert oder gelöscht werden können.

In diesem Falle wird das Attribut auf `atomicselection="true"` gesetzt. Der (Default-)Wert `false` lässt zu, dass wie sonst auch beliebige Fragmente des Elementinhalts selektiert werden können.

Anmerkung

Das Attribut ist nur im Design Mode von MSHTML bei zusätzlich aktiviertem Atomic Selection Mode wirksam. (Hierfür muss das Kommando IDM_ATOMICSELECTION ausgeführt worden sein. Der Aufruf ist nicht scriptbar, kann also nicht aus dem Dokument selbst getätigt werden.)

Anwendbar auf

Alle Elemente mit Ausnahme von `<applet>`, `<base/>`, `<basefont/>`, `<html>`, `<link/>`, `<meta/>`, `<noscript>`, `<script>`, `<style>`, `<title>`

contenteditable

Syntax

contenteditable = "inherit" (Default) | "false" | "true"

Plattform

MSHTML Editor (Internet Explorer ab 5.5)

Beschreibung

Das Attribut `contenteditable` kann für alle Elemente mit selektierbarem Textinhalt eingesetzt werden. Das Attribut erlaubt oder untersagt (Default) das Ändern des Elementinhalts durch Überschreiben, Löschen oder Hinzufügen.

Als einziges der Edit-Attribute ist `contenteditable` auch im normalen Browsermodus aktiv – entsprechend mit `"true"` gekennzeichnete Elemente können also im Browserfenster editiert werden. Die Änderungen sind nicht persistent, werden also beim Neuladen zurückgenommen – geänderte Inhalte lassen sich jedoch in dieser Form ausdrucken.

Anmerkung

Das Attribut `contenteditable` wird auch in der Working Draft der WHAT WG zum HTML-Nachfolger »Web Applications 1.0« als Standardattribut vorgeschlagen. Zugelassen sind ausschließlich `"true"` und `"false"` als Attributwerte (wobei der leere String ebenfalls als `"true"` gewertet wird).[5]

Anwendbar auf

alle Elemente mit Ausnahme von `<applet>`, `<area/>`, `<base/>`, `<basefont/>`, `<caption>`, `<col/>`, `<colgroup>`, `<embed>`, `<frame/>`, `<frameset>`, `<html>`, `<iframe>`, `<img/>`, `<link/>`, `<meta/>`, `<nextid/>`, `<noscript>`, `<script>`, `<style>`, `<table>`, `<tfoot>`, `<thead>`, `<td>`, `<title>`, `<th>`, `<tr>`, `<select>`

5 Siehe: http://whatwg.org/specs/web-apps/current-work/#contenteditable0.

unselectable

Syntax
unselectable = "on" | "off" (Default)
Plattform
MSHTML Editor (Internet Explorer ab 5.5)
Beschreibung
Das Attribut `unselectable` wird denjenigen Elementen beigegeben, die nicht während des Editiervorgangs ausgewählt werden können sollen. Es soll damit verhindert werden, dass die entsprechenden Inhalte geändert oder gelöscht werden können.
Anwendbar auf
Alle Elemente **mit Ausnahme von** `<base/>`, `<basefont/>`, `<html>`, `<link/>`, `<meta/>`, `<noscript>`, `<title>`, `<script>`, `<style>`

Dieses Kapitel enthält eine kommentierte und erläuterte Referenz aller für HTML und XHTML definierten Ersetzungsentitäten und ihrer korrespondierenden Character-Entitäten, eingeteilt in funktionale Gruppen.

16 Referenz HTML Entities

Entities oder Entitäten werden in HTML-Dokumenten dort benötigt, wo bestimmte Sonderzeichen oder Glyphen benötigt werden, die nicht zum Zeichensatz des darzustellenden Dokuments gehören. Auf den folgenden Seiten erhalten Sie eine vollständige Übersicht.

16.1 HTML-Entities und Character-Entities

HTML-Steuerzeichen

Genau wie in SGML und XML existieren in HTML eine Reihe vordefinierter Entitäten zur Umschreibung der Markup-Steuerzeichen. Dies ist erforderlich, wenn öffnende oder schließende Tagklammern oder das Ampersand[1] literal im Text auftreten sollen, beispielsweise bei der Darstellung von Codeabschnitten.

Beachten Sie, dass das **einfache Anführungszeichen** nicht als HTML-Entity existiert (im Gegensatz zu XML, wo es zu den vordefinierten Entities zählt).

Zeichen	HTML-Entity	Character-Entity	Erläuterung
<	<	<	öffnende Tagklammer
>	>	>	schließende Tagklammer
&	&	&	Ampersand (Entities)
'	– [']	'	Apostroph (als Stringbegrenzer)
"	"	"	doppelte Anführungszeichen (als Stringbegrenzer)

Tabelle 16.1 Entities zur Umschreibung von HTML-Steuerzeichen

1 Das Ampersand ist unkritisch, wenn ihm ein Leerzeichen folgt.

Kein Entity für das einfache Anführungszeichen

Das Entity `'` ist kein HTML-Standard, sondern stammt aus XML. Es funktioniert nicht im Internet Explorer. Verwenden Sie für den Apostroph zur Sicherheit das Character-Entity `'`.

Kommerzielle Symbolzeichen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
@	–	@	At-Zeichen
©	©	©	Copyrightzeichen
®	®	®	Registrierter Name
™	™	™	eingetragenes Warenzeichen

Tabelle 16.2 Entities für kommerzielle Symbolzeichen

Unter den Symbolzeichen besitzt das @ als Teil einer E-Mail-Adresse besondere Bedeutung. Ein eigenes HTML-Entity existiert hierfür zwar nicht, jedoch kann das Character-Entity `@` eingesetzt werden, um eine E-Mail-Adresse im Seitenquellcode zu verschleiern (auch wenn dies heutzutage kein großes Hindernis mehr darstellt). Der Mailclient setzt es wieder in das @-Zeichen um und wird dadurch nicht behindert.

Akzentzeichen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
`	–	`	Gravis
´	´	´	Akut
~	–	~	Tilde
˜	˜	˜	kleine Tilde
^	–	^	Zirkumflex
ˆ	ˆ	ˆ	kleiner Zirkumflex
¨	¨	¨	Trema
¸	¸	¸	Cedille (untergesetzt)

Tabelle 16.3 Entities für Akzentzeichen (diakritische Modifizierer)

Beachten Sie den Unterschied zwischen Zirkumflex `ˆ` und Tilde `˜` als diakritische Zeichen einerseits und dem »großen« Zirkumflex, der beispielsweise in regulären Ausdrücken eingesetzt wird, und der einzeln stehenden Tilde andererseits. Die Akzentzeichen Akut und Gravis (Tic und Back-Tic) sollten keinesfalls als Stringbegrenzer eingesetzt werden.

Großbuchstaben mit Akzentzeichen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
À	À	À	A gravis
Á	Á	Á	A akut
Â	Â	Â	A mit Zirkumflex
Ã	Ã	Ã	A mit Tilde
Ä	Ä	Ä	Ä
Å	Å	Å	A mit Ring
Ç	Ç	Ç	C mit Cedille
È	È	È	E gravis
É	É	É	E akut
Ê	Ê	Ê	E mit Zirkumflex
Ë	Ë	Ë	E trema
Ì	Ì	Ì	I gravis
Í	Í	Í	I akut
Î	Î	Î	I mit Zirkumflex
Ï	Ï	Ï	I trema
Ñ	Ñ	Ñ	N mit Tilde
Ò	Ò	Ò	O gravis
Ó	Ó	Ó	O akut
Ô	Ô	Ô	O mit Zirkumflex
Õ	Õ	Õ	O mit Tilde
Ö	Ö	Ö	Ö
Ø	Ø	Ø	O mit Schrägstrich
Š	Š	Š	S mit Caron
Ù	Ù	Ù	U gravis
Ú	Ú	Ú	U akut
Û	Û	Û	U mit Zirkumflex
Ü	Ü	Ü	Ü
Ý	Ý	Ý	Y akut
Ÿ	Ÿ	Ÿ	Y trema

Tabelle 16.4 Entities für Großbuchstaben mit Akzentzeichen (diakritische Zeichen)

Die Zeichen Š und Ÿ entstammen dem Set »Latin Extended-A«. HTML-Entities für diakritische türkische Buchstaben (z. B. İ oder ı) fehlen – diese müssen durch Character-Entities umschrieben werden. Eine Browserunterstützung sollte in diesem Fall getestet werden.

Kleinbuchstaben mit Akzentzeichen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
à	à	à	a gravis
á	á	á	a akut
â	â	â	a mit Zirkumflex
ã	ã	ã	a mit Tilde
ä	ä	ä	ä
å	å	å	a mit Ring
ç	ç	ç	c mit Cedille
è	è	è	e gravis
é	é	é	e akut
ê	ê	ê	e mit Zirkumflex
ë	ë	ë	e trema
ì	ì	ì	i gravis
í	í	í	i akut
î	î	î	i mit Zirkumflex
ï	ï	ï	i trema
ñ	ñ	ñ	n mit Tilde
ò	ò	ò	o gravis
ó	ó	ó	o akut
ô	ô	ô	o mit Zirkumflex
õ	õ	õ	o mit Tilde
ö	ö	ö	ö
ø	ø	ø	o mit Schrägstrich
š	š	š	s mit Caron
ù	ù	ù	u gravis
ú	ú	ú	u akut
û	û	û	u mit Zirkumflex
ü	ü	ü	ü
ý	ý	ý	y akut
ÿ	ÿ	ÿ	y trema

Tabelle 16.5 Entities für Kleinbuchstaben mit Akzentzeichen (diakritische Zeichen)

Anmerkung: Das Zeichen š entstammt dem Set »Latin Extended-A«.

Ligaturen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
Æ	Æ	Æ	Ligatur AE
æ	æ	æ	Ligatur ae
ß	ß	ß	scharfes »s« (Ligatur sz)
Œ	Œ	Œ	Ligature OE
œ	œ	œ	Ligature oe

Tabelle 16.6 Entities für Ligaturzeichen

Die Ligaturen Œ und œ entstammen dem Set »Latin Extended-A«.

Griechische Großbuchstaben

Beachten Sie, dass die griechischen Großbuchstaben in den Fällen, wo die gleiche **Glyphe** (äußere Form) wie für lateinische Großbuchstaben verwendet wird, nicht mit diesen identisch sind. So haben A (großes A) und Α (großes Alpha) zwar das gleiche Aussehen, aber verschiedene Unicodewerte – ein Stringvergleich zwischen beiden schlüge daher fehl.

Zeichen	HTML-Entity	Character-Entity	Erläuterung
Α	Α	Α	Alpha
Β	Β	Β	Beta
Γ	Γ	Γ	Gamma
Δ	Δ	Δ	Delta
Ε	Ε	Ε	Epsilon
Ζ	Ζ	Ζ	Zeta
Η	Η	Η	Eta
Θ	Θ	Θ	Theta
Ι	Ι	Ι	Iota
Κ	Κ	Κ	Kappa
Λ	Λ	Λ	Lambda
Μ	Μ	Μ	My
Ν	Ν	Ν	Ny
Ξ	Ξ	Ξ	Xi
Ο	Ο	Ο	Omicron
Π	Π	Π	Pi
Ρ	Ρ	Ρ	Rho

Tabelle 16.7 Entities für griechische Großbuchstaben

Zeichen	HTML-Entity	Character-Entity	Erläuterung
Σ	Σ	Σ	Sigma
Τ	Τ	Τ	Tau
Υ	Υ	Υ	Ypsilon
Φ	Φ	Φ	Phi
Χ	Χ	Χ	Chi
Ψ	Ψ	Ψ	Psi
Ω	Ω	Ω	Omega

Tabelle 16.7 Entities für griechische Großbuchstaben (Forts.)

Griechische Kleinbuchstaben

Zeichen	HTML-Entity	Character-Entity	Erläuterung
α	α	α	alpha
β	β	β	beta
γ	γ	γ	gamma
δ	δ	δ	delta
ε	ε	ε	epsilon
ζ	ζ	ζ	zeta
η	η	η	eta
θ	θ	θ	theta
ι	ι	ι	iota
κ	κ	κ	kappa
λ	λ	λ	lambda
μ	μ	μ	my
ν	ν	ν	ny
ξ	ξ	ξ	xi
ο	ο	ο	omicron
π	π	π	pi
ρ	ρ	ρ	rho
ς	ς	ς	sigma (Wortende)
σ	σ	σ	sigma (Wortinneres)
τ	τ	τ	tau
υ	υ	υ	upsilon
φ	φ	φ	phi
χ	χ	χ	chi
ψ	ψ	ψ	psi

Tabelle 16.8 Entities für griechische Kleinbuchstaben

Zeichen	HTML-Entity	Character-Entity	Erläuterung
ω	ω	ω	omega

Tabelle 16.8 Entities für griechische Kleinbuchstaben (Forts.)

Beachten Sie, dass die Glyphen mancher griechischer Buchstaben im mathematischen Kontext als Symbole verwendet werden – mit anderem Unicode-Wert. (Normalerweise sollte dies jedoch keine kritische Fehlerquelle sein – es sei denn für Stringvergleiche.)

Internationale Buchstabenformen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
Ð	&Eth;	Ð	Eth (Island)
ð	ð	ð	eth (Island)
Þ	&Thorn;	Þ	Thorn (Island)
þ	þ	þ	thorn (Island)

Tabelle 16.9 Entities für internationale Buchstaben (Isländisch)

Die Buchstaben Eth und Thorn stammen aus dem Altenglischen bzw. Isländischen. Ihr Lautwert entspricht in etwa dem stimmhaften bzw. dem stimmlosen englischen »th«. Das kleine eth wird auch in der internationalen Lautschrift verwendet.

Interpunktionszeichen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
!	–	!	Ausrufezeichen
?	–	?	Fragezeichen
,	–	,	Komma
.	–	.	Punkt
:	–	:	Doppelpunkt
;	–	;	Semikolon
¡	¡	¡	Invertiertes Ausrufezeichen
¿	¿	¿	Invertiertes Fragezeichen
·	·	·	Gregorianisches Komma

Tabelle 16.10 Entities für Interpunktionszeichen

Die Interpunktionszeichen werden in der Regel literal verwendet, können aber im Bedarfsfall mit Character-Entities umschrieben werden. Für die gebräuchlichen Interpunktionszeichen existieren keine HTML-Entities.

Mathematische Grundsymbole

Zeichen	HTML-Entity	Character-Entity	Erläuterung
=	–	=	Gleichheitszeichen
+	–	+	Pluszeichen
×	×	×	Multiplikationszeichen
÷	÷	÷	Divisionszeichen
%	–	%	Prozentzeichen
‰	‰	‰	Promillezeichen
<	<	<	kleiner als
>	>	>	größer als
¬	¬	¬	nicht (Negation)
±	±	±	Plusminuszeichen
/	⁄	⁄	Bruchteil

Tabelle 16.11 Entities für mathematische Grundsymbole

Anmerkung: Das Symbol × kann auch für das kartesische Produkt stehen.

Mathematische Symbole der höheren Mathematik

Zeichen	HTML-Entity	Character-Entity	Erläuterung
∀	∀	∀	»für alle«
∂	∂	∂	part. Differential
∃	∃	∃	»es existiert ein«
Ø	∅	∅	leere Menge
∇	∇	∇	Nabla-Operator
∈	∈	∈	»Element von«
∉	∉	∉	»kein Element von«
∋	∋	∋	»enthält«
∏	∏	∏	Produkt über n
∑	∑	∑	Summe über n
–	−	−	Minuszeichen
*	∗	∗	Asterisk Operator
√	√	√	Wurzel
∝	∝	∝	»proportional zu«
∞	∞	∞	Unendlich
∠	∠	∠	Winkel
∧	∧	∧	logisches Und

Tabelle 16.12 Entities für Symbole der höheren Mathematik

Zeichen	HTML-Entity	Character-Entity	Erläuterung
∨	∨	∨	logisches Oder
∩	∩	∩	Schnittmenge
∪	∪	∪	Vereinigung
∫	∫	∫	Integral
∴	∴	∴	»daraus folgt«
∼	∼	∼	ähnlich
≅	≅	≅	ungefähr gleich
≈	≈	≈	fast gleich
≠	≠	≠	ungleich
≡	≡	≡	identisch
≤	≤	≤	kleiner oder gleich
≥	≥	≥	größer oder gleich
⊂	⊂	⊂	Teilmenge
⊃	⊃	⊃	Obermenge
⊄	⊄	⊄	nicht Teilmenge
⊆	⊆	⊆	Teilmenge oder gleich
⊇	⊇	⊇	Obermenge oder gleich
⊕	⊕	⊕	Summe
⊗	⊗	⊗	Vektorprodukt
⊥	⊥	⊥	»senkrecht auf«
·	⋅	⋅	Punktoperator

Tabelle 16.12 Entities für Symbole der höheren Mathematik (Forts.)

Typographische Zeichen

Zeichen	HTML-Entity	Character-Entity	Erläuterung
'	– ['[2]]	'	Apostroph
‘	‘	‘	einzelnes linkes Anführungszeichen
’	’	’	einzelnes rechtes Anführungszeichen
‚	‚	‚	einz. linkes Anführungszeichen, unten
"	"	"	doppelte Anführungszeichen
“	“	“	doppelte obere linke Anführungszeichen

Tabelle 16.13 Entities für Anführungszeichen und weitere typographische Symbole

2 **Warnung:** Das Entity `'` ist kein HTML-Standard, sondern stammt aus XML. Es funktioniert nicht im Internet Explorer.

Zeichen	HTML-Entity	Character-Entity	Erläuterung
”	”	”	doppelte rechte Anführungszeichen
„	„	„	doppelte untere linke Anführungszeichen
‹	‹	‹	einfaches französisches Anführungszeichen links
›	›	›	einfaches französisches Anführungszeichen rechts
«	«	«	französische Anführungszeichen links
»	»	»	französische Anführungszeichen rechts
(	–	(	runde Klammer links
)	–	)	runde Klammer rechts
[	–	[	eckige Klammer links
]	–	]	eckige Klammer rechts
{	–	{	geschweifte Klammer links
}	–	}	geschweifte Klammer rechts
&	&	&	Ampersand
*	–	*	Asterisk
/	–	/	Solidus (Slash)
\	–	\	umgekehrter Solidus
-	–	-	Bindestrich (Divis)
-			bedingter Trennstrich
–	–	–	Halbgeviertstrich
—	—	—	Geviertstrich
_	–	_	Unter-, Grundstrich
¯	¯	¯	Makron (»quer«)
‾	‾	‾	Überstrich
|	–			vertikale Linie (Pipe)
¦	¦	¦	gebrochene vertikale Linie
§	§	§	Paragraph
¶	¶	¶	Pilcrow-Zeichen
•	•	•	Gliederungspunkt
…	…	…	Ellipse
◊	◊	◊	Rhombus
†	†	†	Kreuz
‡	‡	‡	Doppelkreuz

Tabelle 16.13 Entities für Anführungszeichen und weitere typographische Symbole (Forts.)

Symbole in Buchstabenform

Zeichen	HTML-Entity	Character-Entity	Erläuterung
µ	µ	µ	Micro, Permeabilität
ϑ	ϑ	ϑ	Theta
ϒ	ϒ	ϒ	Ypsilon
ϖ	ϖ	ϖ	Pi (Kreiszahl)
ℵ	ℵ	ℵ	Alephsymbol[3]
℘	℘	℘	Weierstraß'sches P[4]
℘	ℑ	ℑ	Imaginärteil
ℜ	ℜ	ℜ	Realteil

Tabelle 16.14 Entities für Symbole in Buchstabenform

Beachten Sie, dass `µ` und `μ` zwar die gleiche Glyphe »µ« verwenden, jedoch verschiedene Bedeutung und Unicodewerte besitzen.

Pfeile und weitere Symbole

Zeichen	HTML-Entity	Character-Entity	Erläuterung
	⌈	⌈	Ecke oben links
	⌉	⌉	Ecke oben rechts
	⌊	⌊	Ecke unten links
	⌋	⌋	Ecke unten rechts
←	←	←	Pfeil nach links
↑	↑	↑	Pfeil nach oben
→	→	→	Pfeil nach rechts
↓	↓	↓	Pfeil nach unten
↔	↔	↔	Pfeil links-rechts
↵	↵	↵	Enter
⇐	⇐	⇐	Doppelpfeil n. links
⇑	⇑	⇑	Doppelpfeil n. oben
⇒	⇒	⇒	Doppelpfeil n. rechts
⇓	⇓	⇓	Doppelpfeil n. unten
⇔	⇔	⇔	Doppelpfeil r-l
♠	♠	♠	Pik

Tabelle 16.15 Entities für Pfeile und weitere Symbole

3 Nach Georg Cantor; erste überabzählbare Unendlichkeit (kleinste unendliche Kardinalzahl).

4 Karl Th. Wilhelm Weierstraß (1815–1897), deutscher Mathematiker; Wegbereiter der Analysis.

Zeichen	HTML-Entity	Character-Entity	Erläuterung
♣	♣	♣	Kreuz
♥	♥	♥	Herz
♦	♦	♦	Karo

Tabelle 16.15 Entities für Pfeile und weitere Symbole (Forts.)

Währungssymbole

Zeichen	HTML-Entity	Character-Entity	Erläuterung
$	–	$	Dollarzeichen
¢	¢	¢	Centzeichen
£	£	£	Pfundzeichen
¥	¥	¥	Yenzeichen
ƒ	ƒ	ƒ	Florin
€	€[5]	€	Eurozeichen
¤	¤	¤	Allgemeines Währungssymbol

Tabelle 16.16 Entities für Währungssymbole

Das Dollarzeichen kann problemlos in normalem HTML-Text oder innerhalb von Attributwerten verwendet werden.

Whitespace und Zwischenräume

Zeichen	HTML-Entity	Character-Entity	Erläuterung
(unsichtbar)	–		Leerzeichen (Viertelgeviert)
(unsichtbar)			geschütztes Leerzeichen (Viertelgeviert)
(unsichtbar)			en-Zwischenraum (Halbgeviert)
(unsichtbar)			em-Zwischenraum (Geviert)
(unsichtbar)			dünner Zwischenraum (Achtelgeviert)
(unsichtbar)	‌	‌	trennendes Nullbreitenzeichen
(unsichtbar)	‍	‍	verbindendes Nullbreitenzeichen
(unsichtbar)	‎	‎	Textrichtungs-markierung »ltr«
(unsichtbar)	‏	‏	Textrichtungs-markierung »rtl«

Tabelle 16.17 Entities für Leerraum und Spationierung

5 Das Euro-Symbol ist Teil des Zeichensatzes ISO-8859–15, wird daher anzeigeseitig nicht von allen Browsern unterstützt.

Der dünne Zwischenraum ` ` (Achtelgeviert, Spatium) wird (typographisch korrekt) beispielsweise bei der Gruppierung von Telefonnummern eingesetzt; seine Breite entspricht der halben Breite eines regulären Leerzeichens, zwei nebeneinander gestellte ` ` entsprechen also einem ` `, haben aber nicht dessen klammernde Wirkung. Der en-Zwischenraum ` ` (Halbgeviert) ist doppelt so breit wie ein normales Leerzeichen (Viertelgeviert), der em-Zwischenraum ` ` (Geviert) viermal.

Vier weitere der oben aufgeführten Entities stehen in Zusammenhang mit der Internationalisierung von Textdarstellung:

Das trennende Nullbreitenzeichen (Pseudoleerzeichen) `‌` wird beispielsweise im Arabischen als optisch nicht sichtbare Trennung zwischen einzelnen, separat zu verstehenden Wortbestandteilen benötigt. Im gleichen Kontext dient das verbindende Nullbreitenzeichen `‍` zum Zusammenfügen zweier Wortpartikel.

Die Textrichtungsmarker-Entities `‏` und `‎` sollen in Verbindung mit dem `<bdo>`-Element Verwendung finden, werden aber derzeit noch nicht breit unterstützt. Sie sind ebenfalls Nullbreitenzeichen und werden einem ansonsten richtungsneutralen Zeichen (z. B. einem Interpunktionszeichen) vorangestellt, um dessen Leserichtung zu verdeutlichen – dies entscheidet unter Umständen die Position des so markierten Zeichens im dargestellten Text, falls diese nicht aus dem Kontext zu ersehen ist.

Zahlen und Zahlsymbole

Zeichen	HTML-Entity	Character-Entity	Erläuterung
¹	¹	¹	hochgestellte 1
²	²	²	hochgestellte 2
³	³	³	hochgestellte 3
¼	¼	¼	Bruch Viertel
½	½	½	Bruch Halbe
¾	¾	¾	Bruch Dreiviertel
#	–	#	Zahlzeichen, »Hashmark«
ª	ª	ª	weibliches Ordinal
º	º	º	männliches Ordinal
°	°	°	Gradzeichen
′	′	′	Minuten/Fuß
″	″	″	Sekunden/Zoll

Tabelle 16.18 Entities für Zahlen und Zahlensymbole

Die Hochstellungsentities `¹` erscheinen zwar als praktische Abkürzungen für den `<sup>`-Container, werden jedoch kleiner dargestellt. Für Verwechslungen anfällig sind `º` und `°`, die beinahe die gleiche Glyphe verwenden, sowie die Fuß- und Zoll-Symbole `′` und `″`, die kursiv gesetzten einfachen oder doppelten Anführungszeichen ähneln:

Abbildung 16.1 Vergleich von Entity-Glyphen

Character-Entities für alphanumerische Zeichen

Ein Umschreiben alphanumerischer Zeichen durch Character-Entities ist im Allgemeinen nicht erforderlich, da diese stets unmittelbar eingesetzt werden können.

Zeichen	Character-Entity	Zeichen	Character-Entity
0	0	5	5
1	1	6	6
2	2	7	7
3	3	8	8
4	4	9	9

Tabelle 16.19 Umschreibung von Ziffern durch Character-Entities

Zeichen	Character-Entity	Zeichen	Character-Entity
A	A	a	a
B	B	b	b
C	C	c	c
D	D	d	d
E	E	e	e
F	F	f	f
G	G	g	g
H	H	h	h
I	I	i	i

Tabelle 16.20 Umschreibung von Buchstaben durch Character-Entities

Zeichen	Character-Entity	Zeichen	Character-Entity
J	J	j	j
K	K	k	k
L	L	l	l
M	M	m	m
N	N	n	n
O	O	o	o
P	P	p	p
Q	Q	q	q
R	R	r	r
S	S	s	s
T	T	t	t
U	U	u	u
V	&#86	v	v
W	W	w	w
X	X	x	x
Y	Y	y	y
Z	Z	z	z

Tabelle 16.20 Umschreibung von Buchstaben durch Character-Entities (Forts.)

Eine denkbare Einsatzmöglichkeit besteht beim Tarnen von E-Mail-Adressen. So kann der verräterische String `mailto:` mit Character-Entities maskiert werden:

```
    m    a    i    l    t    o    :

&#109;&#97;&#105;&#108;&#116;&#111;&#58;
```

wobei hier für den Doppelpunkt das Character-Entity `:` eingesetzt wurde. Maskiert man zusätzlich das @ mit `@`, erzielt man folgendes Ergebnis:

```
<a href="&#109;&#97;&#105;&#108;&#116;&#111;&#58;
ich&#64;meine-firma.de">Mail</a>
```

Ein Mailclient lässt sich durch diese Umschreibungen nicht irritieren, das mechanische Auslesen von Mail-Adressen aus dem Quelltext wird zumindest erschwert.[6]

Anmerkung: Man kann die Zeichen der eigentlichen Adresse auch durch Hex-URL-Encoding maskieren (z. B. das Zeichen »a« durch `%61`) – der String `mailto:`, der das Protokoll nennt, muss jedoch entweder im Klartext oder wie oben Character-encoded stehen.

6 Man kann davon ausgehen, dass dieser Trick bekannt ist, aber dennoch eine Hürde darstellt.

Dieses Kapitel enthält eine Referenz der für URI-Strings definierten URL-Kodierungen nach RFC 2396 und der Sprachkurzbezeichner (Language Codes) nach RFC 1766.

17 URL-Encoding und Language-Codes

Innerhalb von HTML-Dokumenten treffen auf die Werte bestimmter Attributtypen gewisse Einschränkungen zu. Dies betrifft zum Einen alle Attribute, deren Werte auf URI-Strings beruhen, für die also die Regeln des URL-Encoding beachtet werden müssen. Die zweite betroffene Attributgruppe beschreibt die Sprache eines Elementinhalts. Ihr Inhalt muss einer Sprachangabe gemäß RFC 1766 entsprechen. Beide Sachverhalte sind auf den folgenden Seiten erläutert.

17.1 URL-Encoding

Die HTML-Spezifikation erlaubt die unmittelbare Verwendung des gesamten Zeichensatzes von ISO-Latin, respektive auch den aus Unicode. Diese Freiheit erstreckt sich jedoch nicht auf URI-Strings innerhalb von Dokumenten. Dies liegt daran, dass die Spezifikation von URI-Strings (RFC 2396) lediglich eine Untergruppe des ASCII-Zeichensatzes zulässt – dies sind:

- die alphanumerischen Zeichen a–z, A–Z und 0–9
- die Interpunktionszeichen . – , ' _ ! ( ) ~ * ; ? : / =
- die Sonderzeichen @ & + $

All diese Zeichen dürfen unmittelbar in URI-Strings auftreten. Jedes andere benötigte Zeichen *muss* in jedem Fall gemäß den Regeln der URL-Encodierung maskiert werden. Kritische Zeichen werden durch hexadezimal codierte **Escapesequenzen** `%xx` ersetzt. Allgemein ist es üblich, die hexadezimalen Escapesequenzen in Großbuchstaben zu schreiben.

Escaped werden müssen im URI-String enthaltene Leerzeichen (SPC) und weitere Zeichen, die entweder im URI-String eine Trennfunktion besitzen, eine Fehlerquelle auf Grund zwischen Anwendungen und Plattformen variierender Interpretationen darstellen oder im Kontext der Protokolle von Internet-Gateways verwendet werden.

In der RFC 2396 und der an diese anschließenden RFC 2732 (für IPv6) sind diejenigen Zeichen gelistet, die in Bezug auf URI-Strings gesondert behandelt werden sollten oder unter Umständen maskiert werden müssen. Diese werden als **»reservierte« Zeichen** bezeichnet. Ihnen gegenüber steht eine weitere Gruppe, die »nicht-reservierten« Zeichen.

Nicht-reservierte Zeichen

Zu den nicht-reservierten Zeichen zählen die alphanumerischen Zeichen, also alle Buchstaben und Ziffern des ASCII-Zeichensatzes, sowie die in der folgenden Tabelle aufgeführten Zeichen, die Markierungszeichen (*marks)*.

Die Markierungszeichen

Zeichen	Bedeutung	Unicode	Zeichen	Bedeutung	Unicode
-	Bindestrich	U+002D	~	Tilde	U+007E
_	Unterstrich	U+005F	*	Asterisk	U+002A
.	Punkt	U+002E	(	runde Klammer links	U+0028
'	Apostroph	U+0027	)	runde Klammer rechts	U+0029
!	Ausrufezeichen	U+0021	%	**Prozentzeichen**	U+0025

Die nicht-reservierten Zeichen brauchen im Allgemeinen nicht escaped werden, jedoch ist dies gestattet, ohne dass dabei die Syntax des Links verändert würde.

Anmerkung: Bei vielen Zeichen ist zu beachten, dass es mit oft optisch identischen Glyphen repräsentierte Zeichen abweichender Bedeutung gibt; für den Apostroph (Hochkomma) beispielsweise die linken und rechten einfachen Anführungszeichen U+2018 und U+2019 sowie die einfachen geraden Anführungszeichen U+2032 und U+2033. Ähnliche Zweideutigkeiten existieren für das Bindestrich-Minus (Hyphen).

Reservierte Zeichen (reserved)

Die reservierten Zeichen haben insofern besondere Bedeutung, als dass sie als **Trennzeichen** (*delimiter)* zwischen funktionalen Bestandteilen einer URI fungieren können. So leitet das Fragezeichen einen Querystring ein, der Doppelpunkt einen Portbezeichner. In dieser Bedeutung verwendet dürfen die reservierten Zeichen *nicht* escaped werden.

Die Verwendung der Zeichen in ihrer lexikalischen Bedeutung in Teilen von URI-Strings ist jedoch ebenfalls gestattet (deshalb die Überschneidung mit den zu escapenden Zeichen) – in diesem Fall müssen sie escaped werden.

Die reservierten Zeichen

Zeichen	Bedeutung	Unicode	Zeichen	Bedeutung	Unicode
;	Semikolon	U+003B	+	Plus	U+002B
/	Slash (Solidus)	U+0037	$	Dollarzeichen	U+0024
?	Fragezeichen	U+003F	,	Komma	U+002C
:	Doppelpunkt	U+003A	#	Hash-Zeichen	U+0023
@	At-Zeichen	U+0040	[	**eckige Klammer links**[1]	U+005B
&	Ampersand	U+0026	]	**eckige Klammer rechts**[1]	U+005D
=	Gleichheitszeichen	U+003D			

Verbotene Zeichen

In URI-Strings generell verboten ist eine dritte Gruppe von Zeichen, an erster Stelle die nicht-druckbaren Zeichen (*control characters*), also die ersten 33 ASCII-Zeichen von U+0000 bis U+0032, sowie das Zeichen U+007F (DEL).

Hex Code	ASCII Zeichen	URL Encodierung
0x00 ~ 0x1F		%00 ~ %1F
0x20	Leerzeichen	+ (Plus)
0x21 ~ 0x29	! " # $ % & ' ( )	%21 ~ %29
0x2A	* (Asterisk)	* (unverändert)
0x2B	+ (Plus)	%2B
0x2C	, (Komma)	%2C
0x2D	- (Bindestrich)	- (unverändert)
0x2E	. (Punkt)	. (unverändert)
0x2F	/ (Slash)	%2F
0x30 ~ 0x39	0 ~ 9	0 ~ 9 (unverändert)
0x3A ~ 0x40	: ; < = > ? @	%3A ~ %40
0x41 ~ 0x5A	A ~ Z	A ~ Z (unverändert)
0x5B ~ 0x5E	[\ ] ^	%5B ~ %5E
0x5F	_ (Unterstrich)	_ (unverändert)
0x60	` (Backquote)	%60
0x61 ~ 0x7A	a ~ z	a ~ z (unverändert)
0x7B ~ ... etc.		%7B ~ ... etc.

Tabelle 17.1 Übersicht über die URL-Encoding von ASCII-Zeichen

1 Eckige Klammern sind für IPv6-URIs gedacht. Sie werden ansonsten escaped.

URL-Encodierung kann überall dort erforderlich sein, wo Verweise auf Dateien vorgenommen werden, sei es in Form von Hyperlinks oder Referenzen auf Objekte (Grafiken, Anwendungen). Dies ist in allen Elementen mit `action`-, `href`- oder `src`-Attributen der Fall:

```
<a>, <applet>, <area />, <base />, <body>, <embed>, <form>,
<frame />, <iframe>, <img />, <input type="image" />, <link />,
<object>, <script>
```

17.2 Language Codes nach RFC 1766

Sprachbezeichner (*language codes*) nach RFC 1766 werden in all jenen Attributen benötigt, die Auskunft über die Sprache eines Elementinhalts oder einer externen Ressource geben.

Dies sind die Standardattribute `lang` bzw. `xml:lang` sowie das Attribut `hreflang`, das für die Elemente `<a>` bzw. `<link />` definiert ist.

Anmerkung: Zu zwei Sprachgruppen (Arabisch und Chinesisch) existiert kein übergeordneter Standard, also keine eigene Hauptgruppe. Zu Verwechslungen führen könnte die Abkürzung `be` für Weißrussisch (von »Belarus« – also nicht etwa »Belgian«) und im Gegensatz hierzu `nl-be` (Flämisch) oder `fr-be` (Wallonisch). Eine »einheitliche« belgische Sprache im Sinne des Wortes existiert in der Tat nicht: Die zwei offiziellen Landessprachen Flämisch und Wallonisch sind die der beiden größten Volksgruppen.

Nach RFC 1766 sind 120 verschiedene Sprachbezeichner registriert:

Language Code	Hauptgruppe	Untergruppe
af	Afrikaans	–
ar-ae	Arabisch	Vereinigte Arabische Emirate
ar-bh	Arabisch	Bahrain
ar-dz	Arabisch	Algerien
ar-eg	Arabisch	Ägypten
ar-iq	Arabisch	Irak
ar-jo	Arabisch	Jordanien
ar-kw	Arabisch	Kuwait
ar-lb	Arabisch	Libanon
ar-ly	Arabisch	Libyen
ar-ma	Arabisch	Marokko

Tabelle 17.2 Liste der 120 Sprachbezeichner nach RFC 1766

Language Code	Hauptgruppe	Untergruppe
ar-om	Arabisch	Oman
ar-qa	Arabisch	Qatar
ar-sa	Arabisch	Saudi Arabien
ar-sy	Arabisch	Syrien
ar-tn	Arabisch	Tunesien
ar-ye	Arabisch	Jemen
be	Weißrussisch	–
bg	Bulgarisch	–
ca	Katalanisch	–
cs	Tschechisch	–
da	Dänisch	–
de	Deutsch (Standard)	
de-at	Deutsch	Österreich
de-ch	Deutsch	Schweiz
de-li	Deutsch	Liechtenstein
de-lu	Deutsch	Luxemburg
el	Griechisch	–
en	Englisch (Standard)	
en-au	Englisch	Australien
en-bz	Englisch	Belize
en-ca	Englisch	Kanada
en-gb	Englisch	Großbritannien
en-ie	Englisch	Irland
en-jm	Englisch	Jamaika
en-nz	Englisch	Neuseeland
en-tt	Englisch	Trinidad
en-us	Englisch	USA
en-za	Englisch	Südafrika
es	Spanisch (Standard)	
es-ar	Spanisch	Argentinien
es-bo	Spanisch	Bolivien
es-cl	Spanisch	Chile
es-co	Spanisch	Kolumbien
es-cr	Spanisch	Costa Rica
es-do	Spanisch	Dominikanische Republik

Tabelle 17.2 Liste der 120 Sprachbezeichner nach RFC 1766 (Forts.)

Language Code	Hauptgruppe	Untergruppe
es-ec	Spanisch	Ecuador
es-gt	Spanisch	Guatemala
es-hn	Spanisch	Honduras
es-mx	Spanisch	Mexiko
es-ni	Spanisch	Nicaragua
es-pa	Spanisch	Panama
es-pe	Spanisch	Peru
es-pr	Spanisch	Puerto Rico
es-py	Spanisch	Paraguay
es-sv	Spanisch	El Salvador
es-uy	Spanisch	Uruguay
es-ve	Spanisch	Venezuela
et	Estnisch	–
eu	Baskisch	–
fa	Farsi	–
fi	Finnisch	–
fo	Faeroerisch	–
fr	Französisch (Standard)	
fr-be	Französisch	Belgien
fr-ca	Französisch	Kanada
fr-ch	Französisch	Schweiz
fr-lu	Französisch	Luxemburg
gd	Gälisch (Standard)	Schottland
gd-ie	Gälisch	Irland
he	Hebräisch	–
hi	Hindi	–
hr	Kroatisch	–
hu	Ungarisch	–
in	Indonesisch	–
is	Isländisch	–
it	Italienisch (Standard)	
it-ch	Italienisch	Schweiz
ja	Japanisch	–
ji	Jiddisch	–
ko	Koreanisch	–

Tabelle 17.2 Liste der 120 Sprachbezeichner nach RFC 1766 (Forts.)

Language Code	Hauptgruppe	Untergruppe
lt	Litauisch	–
lv	Lettisch	–
mk	Mazedonisch	–
ms	Malaysisch	–
mt	Maltesisch	–
nl	Holländisch (Standard)	
nl-be	Holländisch	Belgien
no	Norwegisch	–
pl	Polnisch	–
pt	Portugiesisch (Standard)	
pt-br	Portugiesisch	Brasilien
rm	Rätoromanisch	–
ro	Rumänisch (Standard)	
ro-mo	Rumänisch	Moldavien
ru	Russisch (Standard)	
ru-mo	Russisch	Moldavien
sb	Sorbisch	–
sk	Slowakisch	–
sl	Slovenisch	–
sq	Albanisch	–
sr	Serbisch	–
sv	Schwedisch (Standard)	
sv-fi	Schwedisch	Finnland
sx	Sutu	–
sz	Lappländisch	–
th	Thai	–
tn	Tswana	–
tr	Türkisch	–
ts	Tsonga	–
uk	Ukrainisch	–
ur	Urdu	–
ve	Venda	–
vi	Vietnamesisch	–
xh	Xhosa	–
zh-cn	Chinesisch	VR China

Tabelle 17.2 Liste der 120 Sprachbezeichner nach RFC 1766 (Forts.)

Language Code	Hauptgruppe	Untergruppe
zh-hk	Chinesisch	Hong Kong
zh-sg	Chinesisch	Singapur
zh-tw	Chinesisch	Taiwan
zu	Zulu	–

Tabelle 17.2 Liste der 120 Sprachbezeichner nach RFC 1766 (Forts.)

Dieses Kapitel enthält eine Referenz der in HTML und CSS einsetzbaren Klartext-Farbbezeichner (Farbnamen) und der korrespondierenden Farbwerte in RGB-Hexadezimalcode.

18 Farben und Farbwerte

Die Definition von Vorder- und Hintergrundfarben von Elementen zählen zum Bereich der Präsentation. Sie sollten Sie daher vorzugsweise mittels CSS vornehmen, zumal diese Lösung flexibler ist. HTML-Attribute sollten höchstens als Fallbackdefinition für den Fall dienen, dass keine Stylesheetinformationen ausgewertet werden können.

18.1 Farbnamen und Farbwerte

In HTML wie auch in CSS sind sowohl Farbwerte als auch Farbbezeichner in Form vordefinierter Klartextnamen einsetzbar, um Schriftfarbe, Hintergrundfarbe oder Rahmenfarben zu definieren.

Farbzuweisungen in HTML

In **HTML** sind Farben mittels folgender Attribute definierbar:

Attribut	Element
`alink`	`<body>` (aktive Linkfarbe, dokumentweit)
`bgcolor`	`<body>`, `<col>`[2], `<colgroup>`[2], `<table>`, `<tbody>`[2], `<thead>`[2], `<tfoot>`[2], `<tr>`, `<td>`, `<th>`, `<marquee>`[2]
`bordercolor`	`<table>`, `<td>`, `<th>`, `<tr>`, `<frame>`, `<frameset>`, `<iframe>`
`bordercolorlight`[1]	`<table>`, `<td>`, `<th>`, `<tr>`
`bordercolordark`[1]	`<table>`, `<td>`, `<th>`, `<tr>`
`color`	`<font>`, `<basefont>`, `<hr>`[2]
`link`	`<body>` (Linkfarbe, dokumentweit)

Tabelle 18.1 Farbzuweisungen mittels HTML-Attributen

1 Attribut proprietär (IE)
2 Attribut für dieses Element proprietär (IE)

Attribut	Element
text	<body> (Textfarbe, dokumentweit)
vlink	<body> (Farbe besuchter Links, dokumentweit)

Tabelle 18.1 Farbzuweisungen mittels HTML-Attributen (Forts.)

Farbzuweisung mit CSS

Die Zuweisung von Farben an Elemente mittels HTML-Attributen gilt als deprecated. Eine Zuweisung mittels CSS ist vorzuziehen (und in vielen Fällen flexibler).

Farbzuweisungen in CSS

In **CSS** sind Farben mittels folgender Properties definierbar:

- `color` (Vordergrundfarbe eines Elements, d.h. Schriftfarbe)
- `background-color` (Hintergrundfarbe eines Elements)
- `border-color` (Borderfarbe der Box eines Elements; getrennt für alle vier Seiten definierbar)

Anmerkung: In CSS ist anders als in HTML anstelle der Angabe eines Farbwertes in Hexadezimalschreibweise auch eine Angabe in dezimalen RGB-Werten möglich:

```
p { color: #FFA500; }
p { color: rgb(255, 165, 0); }
```

Die dezimalen Werte der RGB-Farbbeschreibung können auch als Prozentwerte ausgedrückt werden (0 entsprechend 0%, 255 entsprechend 100%), wodurch RGB-Farbzuweisungen auch in dieser Form erfolgen können:

```
p {color: rgb(100%,65%,0%); }
```

Neben der ausgeschriebenen Hexadezimalbeschreibung eines Farbwertes ist ebenfalls eine verkürzte Schreibweise erlaubt, sofern die beiden Hexdezimalziffern jedes Farbpaares gleich sind.

Soll also folgende Zuweisung gemacht werden:

```
p {color: #FF9900; }
```

... so kann verkürzend geschrieben werden:

```
p {color: #F90;}
```

Letztens ist ebenfalls die Angabe eines Farbnamens möglich, der jedoch die Angabe eines Hexadezimalwertes oder eines RGB-Wertes vorzuziehen ist:

```
p {color: blue; }
```

Was »browsersafe« bedeutet

Wer früher für Systeme mit 8 Bit Farbtiefe (256 Farben) produzieren musste und dafür die so genannte »CLUT[3]-Palette« einsetzte, ersparte sich schlaflose Nächte. Diese Palette umfasste eine Zusammenstellung von 216 Farbwerten, die in jedem Fall systemübergreifend von allen gängigen Browsern darstellbar war. Von Haus aus arbeiten PC und Macintosh nicht mit der gleichen 8-Bit-Palette. 40 Farben der mit 8 Bit unterscheidbaren Farbwerte waren auf dem jeweils anderen System nicht reproduzierbar, es blieb eine »gemeinsame Zone« von 216 Farben für die sichere Anzeige von Bildern übrig. Betroffen waren nicht nur für Bilder, sondern genauso per HTML oder CSS definierte Hintergrundfarben, Tabellenhintergründe oder Font-Farben.

Bildschirmfarbtiefe

Stellt die Bildschirmfarbtiefe eine geforderte Farbe nicht zur Verfügung, so wird auf den »nächstähnlichen«, im System vorhandenen Farbwert zurückgegriffen.

Dank der heute verbreiteten leistungsfähigeren Grafikkarten stellt sich dieses Problem nur noch verhältnismäßig selten. Dennoch sind Unterschiede in der Farbinterpretation zwischen PC und Mac möglich, was beispielsweise bei der Anpassung von Grafiken an Hintergrundfarben zutage treten kann.

Farbsynthese – Subtraktiv und additiv

Es gibt zwei Arten der Farbsynthese, die sich grundsätzlich unterscheiden:

- die subtraktive Farbsynthese
- die additive Farbsynthese

Die **subtraktive Farbsynthese** befasst sich mit der Mischung von Farbpigmenten: Ein Farbpigment stellt einen Körper dar, der Licht einer bestimmten Wellenlänge (Farbe) absorbiert und das restliche Licht reflektiert.

Bei der Mischung von Farbpigmenten (von denen jedes Licht einer Frequenz schluckt) entnimmt man dem weißen Licht immer mehr Frequenzen, bis schließlich nichts mehr übrig ist (schwarz).

Die Grundfarben der subtraktiven Farbmischung sind **Cyan**, **Magenta** und **Gelb**, die in der Theorie zusammengemischt Schwarz ergeben. In der Praxis, d.h. im Druckbereich nimmt man Schwarz noch als vierte »Kontrastfarbe« hinzu. Daraus ergibt sich das Vierfarbmodell **CMYK** für den Druckbereich. (**C**yan **M**agenta **Y**ellow **B**lacK).

3 CLUT: Color LookUp Table

Die **additive Farbsynthese** trifft man im täglichen Leben weniger offensichtlich an: Hier geht es um die **Mischung von Licht** verschiedener Farbe, wofür ein aktiver Strahler benötigt wird – beispielsweise ein Bildschirm.

Daher spricht man bei additiver Farbsynthese auch von »Bildschirmfarben«. Weißes Licht kann mittels eines Prismas in sein Spektrum zerlegt werden, in umgekehrter Richtung ist ebenso eine Zusammenführung möglich. Aus den drei Grundfarben der additiven Farbmischung **Rot**, **Grün**, **Blau** (das RGB-Farbmodell) erhält man wiederum Weiß. Die Gesamtheit der darstellbaren Farbtöne eines Systems bezeichnet man als »Farbraum« (*colorspace*).

Additive und subtraktive Farbsynthese

Es ist wichtig zu wissen, dass additive und subtraktive Farbsynthese **nicht den gleichen Farbraum** besitzen. Auf dem Bildschirm sind Farben darstellbar, die nicht druckbar sind: Der Farbraum von RGB ist größer als der von CMYK.

RGB-Hexadezimalwerte

Farben in einem HTML-Dokument werden vorzugsweise in (leider etwas unanschaulicher) hexadezimaler Schreibweise `#RRGGBB` angegeben. Die Buchstabenpaare `RR`, `GG` und `BB` stehen hierbei stellvertretend für je eine zweistellige hexadezimale Zahl zwischen `00` (entspricht dezimal 0) und `FF` (entspricht dezimal 255). Jede dieser Hexzahlen steht für einen der drei Farbanteile der additiven Farbsynthese (Grundfarben Rot, Grün, Blau).

Für den Computer entspricht eine zweistellige Hexadezimalzahl genau einem Byte (binär: 8 Bit), so dass für die Farbmischung 3 Byte, also insgesamt 24 Bit zur Verfügung stehen. Rein rechnerisch können damit über 16 Millionen verschiedene Farben aufgeschlüsselt werden.

Klartext-Farbbezeichnungen

Anstelle der Farbangabe im Hexadezimalcode ist es auch möglich, eine Farbe mittels einer (englischen) Klartext-Farbbezeichnung zu wählen, von denen es rund 140 verschiedene gibt. Größtenteils ursprünglich von Netscape implementiert, sind sie inzwischen ebenfalls in SVG 1.0 spezifiziert und als so genannte »X11 Farbnamen« bekannt.

Von den benannten Farben sind nur wenige »browsersafe« (die entsprechenden Farbnamen sind in den folgenden Tabellen **fett** markiert).

Bei 256 Bit Farbtiefe des Displays springt die Anzeige der nicht browsersafen Farben zu vergleichbaren benachbarten Farbwerten. Teilweise gibt es hierbei Unterschiede zwischen der Farbinterpretation zwischen 256 Bit PC und 256 Bit Mac

Plattformen (entsprechende Farbnamen in den folgenden Tabellen mit nachgestelltem * markiert). Benannte Grautöne sind ursprünglich mit »…gray« spezifiziert. Im CSS3 Color Module[4] wird, bei gleich bleibendem Farbwert, stets auch die Schreibweise »…grey« gültig sein (in einigen Fällen auch umgekehrt). In den Tabellen ist diese Schreibweise parallel mit »…gray/…grey« o.ä. angegeben. Verwenden Sie einstweilen jedoch die Bezeichner aus CSS3 mit Vorsicht.

Grundfarben

Die folgenden 16 Farben funktionieren recht zuverlässig auch plattformübergreifend. Es sind die 16 Standardfarben der alten Windows VGA-Farbpalette. Nicht alle von ihnen sind jedoch browsersafe.

Farbname	Hexcode	Farbname	Hexcode
aqua	#00FFFF	navy	#000080
black	#000000	olive	#808000
blue	#0000FF	purple	#800080
fuchsia/magenta	#FF00FF	**red**	#FF0000
gray	#808080	silver	#C0C0C0
green	#008000	teal	#008080
lime	#00FF00	**white**	#FFFFFF
maroon	#800000	**yellow**	#FFFF00

Tabelle 18.2 Grundfarbenpalette

Anmerkung: In der Spezifikation zu CSS 2.1 sind als **offizielle Farbnamen** nur die sechzehn Grundfarben der oben stehenden Tabelle genannt. Darüber hinaus ist der Farbname »orange« (entsprechend #FFA500) spezifiziert, wodurch die Grundfarbenpalette in CSS 2.1 auf siebzehn Farbtöne anwächst.

Neutrale Farbwerte

Neutrale Farbwerte sind Grautöne aller Art von Schwarz bis Weiß. Ein neutraler Grauton zeichnet sich durch jeweils gleichen Wert bei Rot-, Grün- und Blaukomponente aus. Hier die acht benannten Grautöne in alphabetischer Reihenfolge (zwei weitere, als Grautöne benannte Farben, lightslategrey und darkslategrey, sind nicht wirklich neutral):

4 Siehe: http://www.w3.org/TR/css3-color.

Farbname	Hexcode	Farbname	Hexcode
black	#000000	lightgrey/lightgray	#D3D3D3
darkgrey/darkgray	#A9A9A9	silver	#C0C0C0
dimgrey/dimgray	#696969	**white**	#FFFFFF
gainsboro	#DCDCDC	whitesmoke	#F5F5F5
gray/grey	#808080		

Tabelle 18.3 Neutrale Palette

Anmerkung: Beachten Sie, dass die subjektive Wirkung eines Grautons sehr stark von den umgebenden Farben abhängt. In den unten aufgeführten, farbbezogenen Tabellen finden Sie auch »Beinahe-Grautöne« unterschiedlicher Tendenz, die komplementär oder harmonierend zur Farbumgebung eingesetzt werden können.

Helle Farbwerte

Für Dokumenthintergründe sind helle Farbtöne in der Regel gut geeignet, da im Normalfall dunkle Schrift auf hellem Hintergrund den bestmöglichen Kontrast bildet. Die folgenden Farbtöne sind hell bis weiß, jedoch in den meisten Fällen nicht farbneutral. Hier in alphabetischer Reihenfolge:

Farbname	Hexcode	Farbname	Hexcode
azure	#F0FFFF	linen	#FAF0E6
floralwhite	#FFFAF0	mintcream	#F5FFFA
ghostwhite	#F8F8FF	mistyrose	#FFE4E1
honeydew	#F0FFF0	oldlace	#FDF5E6
ivory	#FFF0F0	seashell	#FFF5EE
lavender	#E6E6FA	snow	#FFFAFA
lavenderblush	#FFF0F5	**white**	#FFFFFF
lightcyan	#E0FFFF	whitesmoke	#F5F5F5
lightyellow	#FFFFF0		

Tabelle 18.4 Hintergrundfarbpalette

Anmerkung: Alle hier gelisteten Farbwerte springen auf Systemen mit 256 Bit Farbtiefe zu Weiß um. Darüber hinaus geeignet sind Farbtöne wie aliceblue, blanchedalmond oder cornsilk.

Dunkle Farbwerte

Die folgenden Farbtöne sind dunkel bis schwarz, jedoch in den meisten Fällen nicht farbneutral. Sie eignen sich bei entsprechend gewählter Hintergrundfarbe als Schrift- bzw. Vordergrundfarbe. Hier in alphabetischer Reihenfolge:

Farbname	Hexcode	Farbname	Hexcode
black	#000000	green	#008000
darkblue	#00008B	indigo	#4B0082
darkgreen	#006400	maroon	#800000
darkolivegreen	#556B2F	mediumvioletred	#C71385
darkred	#8B0000	midnightblue	#191970
darkslate-grey/darkslategray[5]	#2F4F4F	navy	#000080
dimgray/dimgrey	#696969	purple	#800080
firebrick[5]	#B22222	seagreen	#2E8B57
gray/grey	#808080	sienna	#A0522D

Tabelle 18.5 Vordergrundfarbpalette

Anmerkung: Bei entsprechend hellem Hintergrund sind auch viele weitere Farben geeignet – die oben stehende Auswahl ist subjektiv.

Grüntöne

Farbname	Hexcode	Farbname	Hexcode
aquamarine	#7FFFD4	limegreen	#32CD32
chartreuse	#7FFF00	mediumaquamarine	#66CDAA
darkgreen	#006400	mediumseagreen	#3CB371
darkolivegreen	#556B2F	mediumspringgreen	#00FA9A
darkseagreen	#8FBC8F	mediumturquoise	#48D1CC
forestgreen	#228B22	olive	#808000
green	#008000	olivedrab	#6B8E23
greenyellow	#ADFF2F	palegreen	#98FB98
honeydew	#F0FFF0	seagreen	#2E8B57
lawngreen	#7CFC00	springgreen	#00FF7F
lightgreen	#90EE90	turquoise[5]	#40E0D0
lightseagreen[5]	#20B2AA	yellowgreen	#9ACD32

Tabelle 18.6 Verschiedene Farbtöne der Grünpalette

5 Differenz Mac/PC bei 8-Bit-Farbtiefe.

Farbname	Hexcode	Farbname	Hexcode
lime	#00FF00		

Tabelle 18.6 Verschiedene Farbtöne der Grünpalette

Gelbtöne

Farbname	Hexcode	Farbname	Hexcode
beige	#F5F5DC	lemonchiffon	#FFFACD
blanchedalmond	#FFEBCD	lightyellow	#FFFFE0
cornsilk	#FFF8DC	moccasin	#FFE4B5
darkgoldenrod	#B8860B	navajowhite	#FFDEAD
darkkhaki	#BDB76B	oldlace	#FDF5E6
floralwhite	#FFFAF0	orange	#FFA500
gold	#FFD700	palegoldenrod	#EEE8AA
goldenrod	#DAA520	papayawhip	#FFEFD5
ivory	#FFFFF0	wheat	#F5DEB3
khaki	#F0E68C	**yellow**	#FFFF00

Tabelle 18.7 Verschiedene Farbtöne der Gelbpalette

Rot- und Brauntöne

Farbname	Hexcode	Farbname	Hexcode
antiquewhite	#FAEBD7	mediumorchid	#BA55D3
bisque	#FFE4C4	mediumvioletred	#C71585
burlywood	#DEB887	mistyrose	#FFE4E1
brown	#A52A2A	orangered	#FF4500
chocolate[6]	#D2691E	orchid	#DA70D6
coral	#FF7F50	palevioletred	#DB7093
crimson	#DC143C	peachpuff	#FFDAB9
darkmagenta	#8B008B	peru	#CD853F
darkorange	#FF8C00	pink	#FFC0CB
darkorchid	#9932CC	plum	#DDA0DD
darksalmon	#E9967A	purple	#800080
darkred	#8B0000	**red**	#FF0000
deeppink	#FF1493	rosybrown	#BC8F8F
firebrick[6]	#B22222	saddlebrown	#8B4513

Tabelle 18.8 Verschiedene Farbtöne der Rot- und Braunpalette

6 Differenz Mac/PC bei 8-Bit-Farbtiefe.

Farbname	Hexcode	Farbname	Hexcode
hotpink	#FF69B4	salmon	#FA8072
indianred	#CD5C5C	sandybrown	#F4A460
lavenderblush	#FFF0F5	seashell	#FFF5EE
lightcoral	#F08080	sienna	#A0522D
lightpink	#FFB6C1	snow	#FFFAFA
lightsalmon	#FFA07A	tan	#D2B48C
linen	#FAF0E6	thistle	#D8BFD8
magenta/fuchsia	#FF00FF	tomato	#FF6347
maroon	#800000		

Tabelle 18.8 Verschiedene Farbtöne der Rot- und Braunpalette

Blautöne

Farbname	Hexcode	Farbname	Hexcode
aliceblue	#F0F8FF	lightblue	#ADD8E6
azure	#F0FFFF	lightcyan[7]	#E0FFFF
blue	#0000FF	lightskyblue	#87CEFA
blueviolet[7]	#8A2BE2	lightsteelblue[7]	#B0C4DE
cadetblue	#5F9EA0	mediumblue	#0000CD
cornflowerblue	#6495ED	mediumpurple	#9370DB
cyan	#00FFFF	mediumslateblue	#7B68EE
darkblue	#00008B	midnightblue	#191970
darkcyan	#008B8B	navy	#000080
darkslateblue	#483D8B	paleturquoise	#AFEEEE
darkslategrey/ darkslategray[7]	#2F4F4F	powderblue	#B0E0E6
deepskyblue	#00BFFF	royalblue[7]	#4169E1
darkturquoise	#00CED1	skyblue	#87CEEB
darkviolet	#9400D3	slateblue	#6A5ACD
dodgerblue[7]	#1E90FF	slategrey/slaytegray	#708090
ghostwhite	#F8F8FF	steelblue	#4682B4
indigo	#4B0082	teal	#008080
lavender	#E6E6FA		

Tabelle 18.9 Verschiedene Farbtöne der Blaupalette

7 Differenz Mac/PC bei 8-Bit-Farbtiefe.

Alle Farbnamen nach CSS 3

Alle Farbnamen gemäß der in CSS 3 spezifizierten X11-Farben und den Namensvarianten »…gray/…grey« gemäß SVG 1 in alphabetischer Reihenfolge:

Farbname	Hexcode	Farbname	Hexcode
aliceblue	#F0F8FF	lightpink	#FFB6C1
antiquewhite	#FAEBD7	lightsalmon	#FFA07A
aqua (cyan)	#00FFFF	lightseagreen	#20B2AA
aquamarine	#7FFFD4	lightskyblue	#87CEFA
azure	#F0FFFF	lightslategray	#778899
beige	#F5F5DC	lightslategrey	#778899
bisque	#FFE4C4	lightsteelblue	#B0C4DE
black	#000000	lightyellow	#FFFFE0
blanchedalmond	#FFEBCD	**lime**	#00FF00
blue	#0000FF	limegreen	#32CD32
blueviolet	#8A2BE2	linen	#FAF0E6
brown	#A52A2A	**magenta** (fuchsia)	#FF00FF
burlywood	#DEB887	maroon	#800000
cadetblue	#5F9EA0	mediumaquamarine	#66CDAA
chartreuse	#7FFF00	mediumblue	#0000CD
chocolate	#D2691E	mediumorchid	#BA55D3
coral	#FF7F50	mediumpurple	#9370DB
cornflowerblue	#6495ED	mediumseagreen	#3CB371
cornsilk	#FFF8DC	mediumslateblue	#7B68EE
crimson	#DC143C	mediumspringgreen	#00FA9A
cyan (aqua)	#00FFFF	mediumturquoise	#48D1CC
darkblue	#00008B	mediumvioletred	#C71585
darkcyan	#008B8B	midnightblue	#191970
darkgoldenrod	#B8860B	mintcream	#F5FFFA
darkgray	#A9A9A9	mistyrose	#FFE4E1
darkgreen	#006400	moccasin	#FFE4B5
darkgrey	#A9A9A9	navajowhite	#FFDEAD
darkkhaki	#BDB76B	navy	#000080
darkmagenta	#8B008B	oldlace	#FDF5E6
darkolivegreen	#556B2F	olive	#808000
darkorange	#FF8C00	olivedrab	#6B8E23
darkorchid	#9932CC	orange	#FFA500

Tabelle 18.10 Farbnamen nach CSS 3 in alphabetischer Reihenfolge

Farbname	Hexcode	Farbname	Hexcode
darkred	#8B0000	orangered	#FF4500
darksalmon	#E9967A	orchid	#DA70D6
darkseagreen	#8FBC8F	palegoldenrod	#EEE8AA
darkslateblue	#483D8B	palegreen	#98FB98
darkslategray	#2F4F4F	paleturquoise	#AFEEEE
darkslategrey	#2F4F4F	palevioletred	#DB7093
darkturquoise	#00CED1	papayawhip	#FFEFD5
darkviolet	#9400D3	peachpuff	#FFDAB9
deeppink	#FF1493	peru	#CD853F
deepskyblue	#00BFFF	pink	#FFC0CB
dimgray	#696969	plum	#DDA0DD
dimgrey	#696969	powderblue	#B0E0E6
dodgerblue	#1E90FF	purple	#800080
firebrick	#B22222	**red**	#FF0000
floralwhite	#FFFAF0	rosybrown	#BC8F8F
forestgreen	#228B22	royalblue	#4169E1
fuchsia (magenta)	#FF00FF	saddlebrown	#8B4513
gainsboro	#DCDCDC	salmon	#FA8072
ghostwhite	#F8F8FF	sandybrown	#F4A460
gold	#FFD700	seagreen	#2E8B57
goldenrod	#DAA520	seashell	#FFF5EE
gray	#808080	sienna	#A0522D
green	#008000	silver	#C0C0C0
greenyellow	#ADFF2F	skyblue	#87CEEB
grey	#808080	slateblue	#6A5ACD
honeydew	#F0FFF0	slategray	#708090
hotpink	#FF69B4	slategrey	#708090
indianred	#CD5C5C	snow	#FFFAFA
indigo	#4B0082	springgreen	#00FF7F
ivory	#FFFFF0	steelblue	#4682B4
khaki	#F0E68C	tan	#D2B48C
lavender	#E6E6FA	teal	#008080
lavenderblush	#FFF0F5	thistle	#D8BFD8
lawngreen	#7CFC00	tomato	#FF6347
lemonchiffon	#FFFACD	turquoise	#40E0D0

Tabelle 18.10 Farbnamen nach CSS 3 in alphabetischer Reihenfolge (Forts.)

Farbname	Hexcode	Farbname	Hexcode
lightblue	#ADD8E6	violet	#EE82EE
lightcoral	#F08080	wheat	#F5DEB3
lightcyan	#E0FFFF	**white**	#FFFFFF
lightgoldenrodyellow	#FAFAD2	whitesmoke	#F5F5F5
lightgray	#D3D3D3	**yellow**	#FFFF00
lightgreen	#90EE90	yellowgreen	#9ACD32
lightgrey	#D3D3D3		

Tabelle 18.10 Farbnamen nach CSS 3 in alphabetischer Reihenfolge (Forts.)

In diesem Kapitel erhalten Sie eine Einführung in die Grundsyntax von CSS 1.0 und 2.0 sowie eine Referenz aller zur Präsentation von HTML-Dokumenten einsetzbaren CSS-Eigenschaften und ihrer Werte.

19 CSS-Referenz

Die Bedeutung von CSS in Zusammenhang mit HTML und XHTML ist fraglos immens – die gesamte Präsentation eines HTML-Dokuments basiert im zeitgenössischen Webdesign auf CSS. Das Layout kann dabei sogar an die jeweilige Zielplattformen angepasst werden, sei dies ein normaler Browser, ein Mobilgerät oder die Ausgabe einer Seite über den Drucker.

19.1 Grundlagen

CSS-Anweisungen definieren die Präsentation des Inhaltes eines HTML-Elementcontainers, wobei sich Präsentationsvorschriften addieren und gegenseitig widerrufen (überschreiben) dürfen sowie aus Quellen mit unterschiedlichen Relationen zum HTML-Dokument zusammengefasst werden können. Die Summe aller auf ein HTML-Dokument anzuwendenden CSS-Regeln wird nach ihrer Zusammenführung und der Regelung möglicher Konflikte als »Stylesheet« bezeichnet.[1]

Die Methode, nach der die Stylesheetregeln addiert werden, wird als »Kaskadierung« bezeichnet. Das Prinzip beruht auf einer Rangeinstufung jeder einzelnen Regel, die sich erstens auf der »Entfernung« der Deklaration der Regel vom Zielelement bezieht und zweitens auf ihre »Passgenauigkeit«, der so genannten Spezifizität.

19.1.1 Formatierungsregeln

Das Stylesheet ist nichts weiter als eine Abfolge von **Formatierungsregeln**, so genannten *rule sets*. Jede Regel zerfällt in zwei Teile, den Selektor und den in geschweifte Klammern eingeschlossenen Deklarationsblock (*declaration-block*), der die eigentlichen Formatierungsanweisungen enthält.

1 Auch eine separate CSS-Datei wird als Stylesheet bezeichnet. Hier ist es jedoch im Sinne eines abstrakten Gesamtregelsatzes zu verstehen.

Die Formatierung muss einem Elementcontainer eindeutig zugewiesen werden können. Diese Zuordnung erfolgt über ein so genanntes Vergleichsmuster – gebildet durch den vorangestellten **Selektor**.

Allgemein kann man eine Formatierungsregel so formulieren:

```
selektor { propertydeklaration }
```

Mehrere Einzelregeln für den gleichen Selektor addieren sich:

```
selektor { propertydeklaration_1 }

selektor { propertydeklaration_2 }
```

… und können zu einem Deklarationsblock zusammengefasst werden. Es dürfen beliebig viele, gleichzeitig gültige Anweisungen auftreten, jeweils durch Semikolon getrennt – man spricht auch von einer so genannten »Properties Collection«:

```
selektor { propertydeklaration_1; propertydeklaration_2 }
```

Lesbarkeit geht vor Platzersparnis – innerhalb der CSS-Syntax spielen Zeilenumbrüche und Leerzeilen keine Rolle – man kann genauso gut schreiben:

```
selektor {
          propertydeklaration_1;
          propertydeklaration_2
          }
```

Jede Eigenschaftsdeklaration ist stets zusammengesetzt aus Property-Bezeichner (*name*) und Property-Wert(en) (*value*) – getrennt durch einen Doppelpunkt. Leerzeichen um den Doppelpunkt sind ebenfalls erlaubt.

```
selektor {property-name : property-value;}
```

Das Ende der Deklaration wird durch ein abschließendes **Semikolon** gekennzeichnet. Ein zugewiesener Wert kann daher selbst auch Leerzeichen enthalten oder die Form einer kommagetrennten Liste annehmen.

Soll nur eine einzige Eigenschaft innerhalb eines Deklarationsblocks festgelegt werden, so kann das Semikolon entfallen – ebenso für das letzte Property vor der schließenden geschweiften Klammer.

Da man eine Regel gelegentlich erweitern muss, ist es jedoch sinnvoll, eine Zuweisung grundsätzlich mit Semikolon zu beenden:

```
selektor {
         property-name_1: property-wert_1;
          property-name_2: property-wert_2;
         }
```

Semikola beachten

Wird ein Semikolon im Inneren eines Blocks vergessen, so wird die entsprechende Zuweisung nicht erkannt und gegebenenfalls auch die auf sie folgenden CSS-Anweisungen nicht ausgeführt.

Reihenfolge der Wertzuweisungen

Wird innerhalb eines Deklarationsblocks einer Eigenschaft mehrfach ein Wert zugewiesen, so gilt die zuletzt geschriebene Anweisung als maßgeblich. Im folgenden Beispiel bekommt die Eigenschaft `property-name_1` zweimal einen Wert zugewiesen, wobei der zweite den ersten überschreibt:

```
selektor {
          property-name_1: property-wert_1;
          property-name_2: property-wert_2;
          property-name_1: property-wert_3;
         }
```

!important-Schalter

Die Anweisung `!important` hinter eine CSS-Regel gesetzt, gibt dieser einen höheren Rang als eine auf das gleiche Element mittels eines Selektors mit identischer Spezifizität zugreifende. Im Konfliktfall wird daher die `!important`-Regel ausgeführt, nicht aber die Regel ohne diesen Schalter. Auch werden die Eigenschaften der so überlagerten Regel nicht zur `!important`-Regel addiert, selbst wenn kein Konflikt vorliegt.

Beispiel: **[zB]**

Bei diesen beiden im Prinzip gleichrangigen Regeln für `<p>`-Elemente wird die Regel ohne `!important`-Schalter *vollständig* ignoriert:

```
/* diese Regel bekommt Vorrang: */
p { border: solid 1px red; padding:5px;} !important

/* folgende Regel wird nicht angewendet: */
p { background-color:#999999; }
```

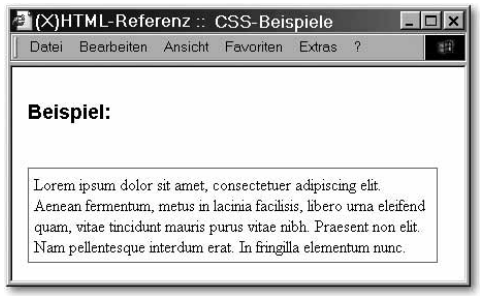

Abbildung 19.1 Die Hintergrundfarbe (Regel 2, s.o.) wird nicht angewendet.

Lässt man den `!important`-Schalter weg und fügt der ersten Regel eine Hintergrundfarbdefinition hinzu, so wird diese durch die Hintergrundfarbe der zweiten Regel überschrieben, da diese als zuletzt stehende ranghöher ist. Dem zweiten Container ist mittels einer Klasse eine andere Hintergrundfarbe zugewiesen:

```
p { border: solid 1px red;
    padding:5px;
    background-color:#ff9999;}
p { background-color:#999999; }
/* "wichtig" mit hellerer Hintergrundfarbe: */
p.wichtig { background-color:#dddddd; }
```

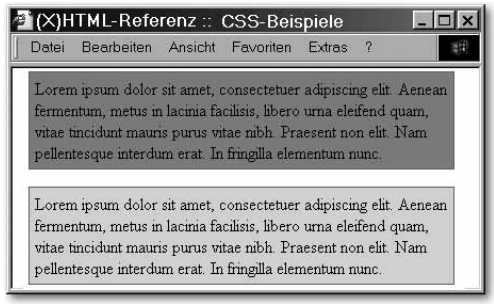

Abbildung 19.2 Ohne !important überwiegt die zweite Regel.

Fügt man den `!important`-Schalter wieder hinzu, so überwiegt erneut die Hintergrundfarbe der ersten Regel. Für den zweiten Container gilt jedoch nach wie vor die durch die Klasse definierte Hintergrundfarbe, da der Klassenselektor gegenüber dem Elementselektor (auch mit `!important`) stets die höhere Spezifizität (Passgenauigkeit) besitzt:

```
p { border: solid 1px red;
    padding:5px;
    background-color:#ff9999;}  !important
```

```
p { background-color:#999999; }
p.wichtig { background-color:#dddddd; }
```

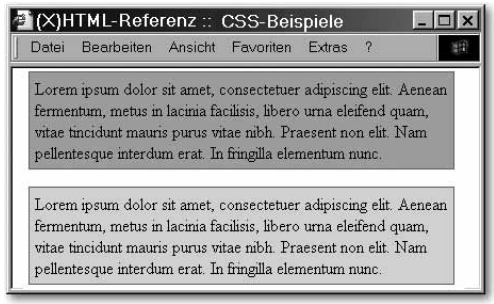

Abbildung 19.3 Die Klasse überschreibt Elementselektor mit !important.

Kompatibilität

Der `!important`-Schalter wird durch *alle* aktuellen Browser unterstützt. Der inzwischen historische Netscape 4.x interpretiert den Befehl nicht.

19.1.2 Position der Styledefinition

Styleanweisungen können zu einem HTML-Element, abhängig vom Ort ihrer Deklaration, in vier hierarchisch unterschiedlich gewichteten Varianten in Beziehung stehen, die im Folgenden eingehender diskutiert werden.

- als Inlinestyle
- als Dokumentstyle
- als externes importiertes Stylesheet
- als externes verlinktes Stylesheet

Inlinestyle

Die Styleanweisung wird mittels des `style`-Attributs unmittelbar im Starttag des zu stylenden Elements eingefügt. Diese Methode steht der Trennung von Präsentation und Dokumentinhalt entgegen und sollte möglichst nicht angewendet werden.

```
...
<p style="font-family:arial">
 Dies wird in Arial dargestellt.</p>
...
```

Der Inlinestyle gilt lokal ausschließlich für den Inhalt des Elements, in dem er definiert ist. Er kann verwendet werden, um gezielt einen globaler gesetzten

Style zu widerrufen. Den gleichen Effekt könnte man (einmalig) mit einem ID-Selektors oder (mehrfach) mit einem Klassenselektor erzielen.

Anmerkung: Ältere Browser waren bei DHTML-Anwendungen mit positionierten Ebenen auf Inlinestyles für Positionsangaben angewiesen – dies ist der einzige Fall, in dem Inlinestyles noch immer erforderlich sein könnten.

Kompatibilität
Inline-Styleanweisungen werden durch *alle* aktuellen Browser unterstützt.

Dokumentstyle

Ein Dokumentstyle wird innerhalb des `<style>`-Containers im Dokumentkopf definiert und gilt global für das gesamte Dokument. Die Trennung zwischen Inhalt und Präsentation ist besser als beim Inlinestyle, die Möglichkeit der unmittelbaren Mehrfachverwendung von Styles für eine größere Zahl von Dokumenten jedoch nicht gegeben.

Aufgrund des sich in den DTD-Definitionen unterscheidenden Inhaltsmodells des `<style>`-Containers muss die Einbindung in HTML und XHTML jeweils unterschiedlich erfolgen.[2]

Das Inhaltsmodell von `<style>` ist in HTML vom Typ `CDATA`. Der Inhalt kann also für alte, nicht stylefähige Browser auskommentiert werden:

```
<style type="text/css">
<!--
        p {font-family:arial}
-->
</style>
```

In XHTML ist das Inhaltsmodell von `<style>` dagegen `#PCDATA`, was die Verwendung von HTML-Kommentaren zum Verstecken des CSS-Inhalts nicht zulässt. Hier muss entweder auf den umgebenden Kommentar verzichtet werden:

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

        p {font-family:arial}

</style>
```

... oder der CSS-Inhalt explizit zum `CDATA`-Bereich erklärt werden:

2 Eine ausführlichere Diskussion der Problematik, insbesondere bei der Auslieferung von XHTML-Dokumenten mit MIME-Typ `"text/html"` anstelle von `"application/xhtml+xml"` finden Sie im HTML-Referenzkapitel Abschnitt `<style>`.

```
<style type="text/css">
<![CDATA[

        p {font-family:arial}

]]>
</style>
```

Kompatibilität

Dokument-Styleanweisungen werden durch *alle* aktuellen Browser unterstützt.

Externes importiertes Stylesheet

Anstatt eine Stylesheet-Datei über einen Link mit dem Dokument zu verknüpfen, kann diese (wirkungsgleich) auch in den Dokumentstyle importiert werden.

Der `@import`-Anweisung wird ein URL übergeben, der zum Basis-URL der aufrufenden Datei aufgelöst wird. Die Syntax lautet wahlweise

```
@import "mein_style.css";
```

oder etwas komplizierter mit dem `url()`-Befehl:

```
@import url("mein_style.css");
```

Besitzt das aufrufende Stylesheet den Basis-URI

```
"http://www.beispiel.de/styles/"
```

... so wird der URI in diesem Fall aufgelöst zu

```
"http://www.beispiel.de/styles/mein_style.css"
```

Eingesetzt wird die `@import`-Anweisung in der Regel im `<style>`-Container. Dies gibt die Möglichkeit, dort auch weitere lokale Styles zu definieren. In diesem Fall muss die `@import`-Anweisung zuerst stehen.

```
<style type="text/css">
   @import url("mein_style.css");
   p {font-family:arial;}
</style>
```

Ebenso ist es möglich, mehrere Stylesheet nacheinander zu importieren:

```
<style type="text/css">
   @import url("mein_style.css");
   @import url("mein_anderer_style.css");
</style>
```

Analog zum `media`-Attribut bei verlinkten Styles ist auch für importierte Styles die Angabe eines oder mehrerer gewünschter Ausgabemedien möglich.

```
<style type="text/css">
   @import url("mein_style.css");
   @import url("screen.css") screen, projection;
   @import url("print.css") print;
</style>
```

Kompatibilität

Der `@import`-Befehl wird lediglich vom historischen Netscape 4.x nicht unterstützt.

Externes verlinktes Stylesheet

Für die Verknüpfung eines Stylesheets mit einem Dokument wird das `<link>`-Element verwendet (im Dokumentkopf):

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

Es können mehrere Stylesheets mit einem Dokument verknüpft werden, wobei diesen auch verschiedene Ausgabemedien zugewiesen werden können. Das `media`-Attribut spielt in diesem Zusammenhang eine wichtige Rolle. Es ist hiermit möglich, auf verschiedene Stylesheets zu verweisen, die dem jeweiligen Ausgabezweck angepasst sind, z. B. mittels `media="handheld"` Styles mit kleinerer Schrift für Mobiltelefone oder PDAs oder andere Farben mittels `media="projection"` für Videobeamer festzulegen oder mit `media="print"` Optionen für Druckausgabe oder mit `media="screen"` solche für Bildschirmausgabe anzugeben.

```
<link rel="stylesheet" type="text/css" href="screen.css"
 media="screen">
<link rel="stylesheet" type="text/css" href="print.css"
 media="print">
```

Der allgemeine Wert `media="all"` kann für die Verlinkung nicht medienspezifischer Styles verwendet werden, die allgemein gültig sein sollen und daher stets anzuwenden sind.

Die `@import`-Anweisung kann auch in einer Stylesheet-Datei auftreten – in diesem Fall muss sie ebenfalls vor den restlichen Styleregeln gesetzt werden:

```
/* Dateiname: screen.css */
   @import url("globale_regeln.css");

/*lokale Regeln: */
   p {font-family:arial;}
```

Die importierten Regeln gelten auch hier, verglichen mit den lokal im Stylesheet deklarierten, als rangniedriger.

Kompatibilität

Externe, verlinkte Stylesheets werden durch alle Browser unterstützt.

19.1.3 Präferenz und Cascading-Prinzip

Im Rahmen des Gesamtstylesheets, das auf das Dokument angewendet wird, addieren sich die CSS-Regeln aller Quellen. Sofern es bei den Eigenschaftszuweisungen keine Konflikte gibt, wird auf einen selektierten Elementcontainer die Summe aller für ihn in Frage kommenden Regeln angewendet. So erhält ein Element mit Klassenattribut zunächst alle Präsentationsvorschriften, die auf seinen Elementtyp zutreffen, und anschließend auch jene, die sich aus der Bindung an eine Klasse ergeben.

Bei einem zusammengesetzten Stylesheet wird der Rang der Regeln nach der Entfernung vom zu stylenden Objekt gewichtet: Je »näher« die Deklaration der Eigenschaft am Element ist, desto ranghöher wird sie eingestuft. Dieses Verhalten wird als »Präferenz« bezeichnet.

- Eine Deklaration im Rahmen eines Inlinestyles ist die ranghöchste und kann nicht überschrieben werden. Sie hat Vorrang vor Dokumentstyles, importierten sowie verlinkten Stylesheets.
- Eine Deklaration im Rahmen eines Dokumentstyles gilt dokumentweit und hat Vorrang vor importierten und verlinkten Stylesheets.
- In einen Dokumentstyleblock importierte Stylesheetregeln haben Vorrang vor verlinkten Stylesheets, sind jedoch rangniedriger als die im Dokumentstylebock definierten Regeln.

Dies liegt daran, dass die Importanweisung zu Beginn des Styleblocks zu erfolgen hat. Die auf die Importanweisung folgenden Regeln gelten als »zuletzt geschrieben« und daher vorrangig. Analog haben die Regeln des mit der letzten Importanweisung importierten Stylesheets Vorrang vor dem zuerst importierten.

- Die Regeln eines verlinkten externen Stylesheetdokuments gelten gegenüber allen anderen als nachrangig.

In ein externes Stylesheet importierte Regeln gelten zwar den Regeln der Stylesheetdatei gegenüber als nachrangig, besitzen gegenüber dem Dokument jedoch den gleichen Rang wie die Regeln der verlinkten Datei (der Konflikt wurde gewissermaßen extern ausgetragen).

Ermittlung der Präferenz in der Präferenzkaskade

Wenn es daran geht festzustellen, welche Regel bzw. welche Eigenschaften welcher Regeln auf ein Element anzuwenden sind, wird folgendes Prinzip angewendet:

1. Die Regel aus der ranghöchsten Quelle gilt.
2. Bei Regeln aus gleichrangigen Quellen gilt die passgenauere, das heißt diejenige mit dem spezifischeren Selektor.

3. Stammen zwei Regeln gleicher Passgenauigkeit aus gleichrangigen Quellen, so gilt die zuletzt geschriebene (d.i. auch die zuletzt importierte, oder die aus der zuletzt verlinkten Quelle).
4. Besitzt bei ansonsten gleichrangigen Regeln eine einen `!important`-Schalter, so wird diese angewendet.

Ist für eine Regel ein **Medientyp** angegeben, so wird sie nur mit Regeln für den gleichen Medientyp verglichen und mit diesen zusammen angewendet.

Ermittlung der Spezifizität der Selektoren

Der Selektor dient der Zuordnung einer CSS-Formatierungsregel zu einem Elementcontainer. Dieser arbeitet dabei anhand eines **Mustervergleichs** (*pattern matching*), wobei jede gefundene Übereinstimmung eines Musters auf ein Element angewendet wird. Dadurch addieren sich gegebenenfalls die Eigenschaftszuweisungen mehrerer, auf denselben Elementcontainer passenden Regeln.

Im Fall eines Konflikts zwischen zugewiesenen Eigenschaften ermittelt sich die dominierende CSS-Eigenschaft danach, wie spezifisch das Muster ihres Selektors formuliert ist.

Rangermittlung gemäß CSS 2.0:

Für einen (einfachen oder beliebig zusammengesetzten) Selektor berechnet die Spezifizität sich wie folgt zu einer dreistelligen Ziffer, indem die Summe der Einzelwertungen gebildet wird:

- Zahl der **ID-Attributselektoren** (multipliziert mit 100)

dazu addiert

- Zahl der **Attributselektoren**, **Klassenselektoren** oder **Pseudoklassen** (multipliziert mit 10)

dazu addiert

- Zahl der **Typselektoren** (multipliziert mit 1)

dazu addiert

- Zahl der **universellen Selektoren** (multipliziert mit 0; universelle Selektoren spielen daher auch in zusammengesetzten Selektoren rangmäßig keine Rolle)

Ein beliebiger Typselektor hat somit den Rang 1, jeder Attributselektor mindestens den Rang 10, ein ID-Selektor mindestens 100. Der universelle Selektor hat allein stehend den Rang 0, ist also stets der rangniedrigste. Sein Auftreten in einem zusammengesetzten Selektor beeinflusst dessen Rang daher nie.

Beispiele: **[zB]**

Selektor	Rang	Begründung
`*`	0	universeller Selektor zählt 0
`p`	1	1 Elementselektor
`p[align]`	11	1 Attribut- + 1 Elementselektor
`p em`	2	2 Elementselektoren
`div *[align]`	11	1 Attribut- + 1 Elementselektor
`div#content p[align]`	112	1 ID- + 1 Attribut- + 2 Elementselektoren

Tabelle 19.1 Beispiele für den Rang von Selektoren gemäß CSS 2.0

Rangermittlung gemäß CSS 2.1:

In CSS 2.1 wird anstelle der dreistelligen Bewertung aus CSS 2.0 die Formulierung der Spezifizität getrennt nach vier, nach Rang geordneten Gruppen A-B-C-D vorgenommen. Hierbei wird pro Gruppe die Zahl der betreffenden Selektoren summiert – es wird also keine übergreifende Gesamtsumme gebildet.

Die formale Einbeziehung von CSS-Regeln des `style`-Attributs hierbei ist neu.

- **Gruppe A:**
  ist 1, wenn die CSS-Regel in einem **style-Attribut** steht, sonst 0
- **Gruppe B:**
  die Zahl der **ID-Attributselektoren**
- **Gruppe C:**
  die Zahl der **Attribut**- und **Klassenselektoren** sowie der **Pseudoklassen**
- **Gruppe D:**
  die Zahl der **Typselektoren**

Der universelle Selektor wird bei dieser Bewertung nicht berücksichtigt.

Eine Bewertung ungleich 0 in einer ranghöheren Gruppe »schlägt« jede beliebige Bewertung einer niedrigeren Gruppe. Die Bewertung `1-0-0-0` ist daher jederzeit ranghöher als `0-99-0-0`, analog `0-0-1-0` ranghöher als `0-0-0-8`.

Beispiele: **[zB]**

Selektor	Rang	Begründung
`*`	0–0–0–0	keine bewertbaren Selektoren
`p`	0–0–0–1	1 Elementselektor
`p[align]`	0–0–1–1	1 Attribut-, 1 Elementselektor

Tabelle 19.2 Beispiele für den Rang von Selektoren gemäß CSS 2.1

Selektor	Rang	Begründung
`p em`	0–0–0–2	2 Elementselektoren
`div *[align]`	0–0–1–1	1 Attribut-, 1 Elementselektor,
`div#content p[align]`	0–1–1–2	1 ID-, 1 Attribut-, 2 Elementselektoren
`style=" ... "`	1–0–0–0	Inlinestyle als am ranghöchsten gewertet

Tabelle 19.2 Beispiele für den Rang von Selektoren gemäß CSS 2.1 (Forts.)

Vorrang vor HTML-Präsentation

HTML-Präsentationsvorschriften in einem Dokument (wie Farbanweisungen, Ausrichtung etc.) werden, soweit sie mit missbilligten Attributen oder Elementen erfolgen, in CSS 2.1 mit dem Rang 0–0–0–0 eingestuft. Eine CSS-Regel hat daher vor ihnen stets Vorrang.

19.1.4 Kommentare in Stylesheets

In CSS-Stylesheets dürfen Kommentare gesetzt werden, die von der Syntax den mehrzeiligen, aus C bekannten Kommentaren entsprechen:

```
/* dies ist ein Kommentar */
```

Innerhalb von Kommentaren sind daher Zeilenumbrüche möglich:

```
/*
dies ist ein Kommentar
*/
```

Keine einzeiligen Kommentare

Einzeilige Kommentare existieren in CSS nicht. Ein Kommentarversuch mit Doppelslash wie in JavaScript führt zu einem Syntaxfehler.

Erklärende Kommentare können am Beginn eines eingebundenen Stylesheets verwendet werden, um dessen Einsatzgebiet zu nennen oder Copyrightvermerke unterzubringen, oder können in längeren Stylesheets als Gliederung oder Erklärung eingesetzt werden.

```
/* lokale Regeln: */
   p   {font-family:arial;
        }
```

Ebenso möglich ist das Auskommentieren einzelner Regeln

```
/* diese Regel ist auskommentiert:
   h1  {font-size:2em;
```

```
        }
*/
```

... oder sogar einzelner Properties einer zusammengesetzten Regel:

```
div { font-family:arial;
      font-size:0.8em;
  /*  font-weight: bold; */
      color: #ff0000;
      }
```

Keine HTML-Kommentare

Der aus HTML bzw. SGML bekannte Kommentar `<!-- ... -->` gilt in CSS nicht. Der einzige Fall, in dem dieser im Kontext mit CSS erscheint, ist im Rahmen eines HTML-Dokuments (d.i. *nicht* eines XHTML-Dokuments!) *außerhalb* der Styledefinitionen unmittelbar innerhalb des Stylecontainers. Letzteres wird durch alle aktuellen Browser unterstützt.

19.2 CSS At-Rules

Die CSS »At-Rules« gelten entweder für das gesamte Stylesheet, um dort bestimmte Voreinstellungen zu treffen, für die keine Selektoren zuständig sind (`@charset`, `@font-face`), oder importieren externe Stylesheetregeln aus einer anderen CSS-Datei (`@import`). Drittens dienen sie zur Formulierung bedingter Abschnitte innerhalb des Stylesheets, die nur unter bestimmten Umständen verwendet werden sollen (`@media` in Abhängigkeit vom Medientyp des Client, `@page` beim Ausdruck des Dokuments). Die At-Rules besitzen daher eine gewisse Sonderstellung gegenüber den restlichen Ausprägungen der CSS-Syntax.

@charset

Gültigkeit

CSS 2, CSS Mobile; (entfällt in CSS 2.1)

Beschreibung

Die Anweisung `@charset` definiert den Zeichensatz des CSS-Dokuments. Dieser wird einfach als Zeichenkette (in Anführungszeichen) übergeben. Eine Bestimmung des Zeichensatzes ist für externe CSS-Dateien unter Umständen von Bedeutung. Ein Dokumentstyle wird mit dem Zeichensatz des beherbergenden HTML-Dokuments interpretiert.

Beispiel

```
@charset "ISO-8859-1";
```

Anmerkung

Diese At-Rule wäre im Wesentlichen erst dann von Interesse, wenn keine westlichen Sprachcodes verwendet werden (kein ISO-Latin).

Kompatibilität

Die `@charset`-Rule wird durch aktuelle Browser **nicht** unterstützt. In CSS 2.1 ist die Regel nicht mehr enthalten.

@font-face

Gültigkeit

CSS 2; (entfällt in CSS 2.1)

Beschreibung

Mittels der Anweisung `@font-face` kann ein benötigter Fontsatz über deren `src`-Property direkt als Datei eingebunden werden. Die Spezifikation nennt hier eine Reihe verschiedener Typen, wobei man in der Praxis allerdings auf die Typen `.eot` (Embedded Open Type, Internet Explorer) und `.prf` (Portable Font Resource, Netscape 4.x) beschränkt ist.

Im folgenden Beispiel wird in `src` mit `local()` zunächst geprüft, ob eine lokale Ressource mit der Bezeichnung „Eurostyle Brock" vorhanden ist, andernfalls soll eine entsprechende Datei geladen werden.

Das Property `font-family` schließlich vergibt einen Bezeichner, mit dem die importierte Schrift anschließend in anderen Regeln eingesetzt werden kann.

Beispiel

```
/* Import einer benötigten Schrift: */
@font-face {
       src: local("Eurostyle BK");
       src: url("eurostyle.eot"), url("eurostyle.pfr");
       font-family: Eurostyle;
}

/* Anwendung der importierten Schrift: */
p          {
            font-family: Eurostyle;
            font-size: 0.8em;
}
```

Kompatibilität

Das `@font-face`-Rule wird durch aktuelle Browser mit Ausnahme des Internet Explorers *nicht* unterstützt, der sich dabei auf das Format EOT (*embedded opentype*) beschränkt. Die Verwendung ist daher als problematisch einzustufen. In CSS 2.1 ist sie nicht mehr enthalten.

@import

Gültigkeit

CSS 1, CSS 2, CSS 2.1, CSS Mobile;

Beschreibung

Die Anweisung `@import` dient dem Importieren externer Stylesheetdateien. Einer externen Stylesheetdatei wird ein URL übergeben, die relativ zum Basis-URI der aufrufenden Datei ausgewertet wird. Neben dem URL kann auch ein bevorzugter Medientyp genannt werden, für den das importierte Stylesheet zu verwenden ist.

In **CSS 1** musste der `@import`-Befehl unter Zuhilfenahme des `url()`-Befehls geschrieben werden:

```
@import url("screen.css");
@import url("print.css") print;
@import url("mobile.css") handheld;
```

In **CSS 2** und **CSS 2.1** kann ein Dateiname auch direkt übergeben werden, wobei dieser in Anführungszeichen gesetzt werden muss. Gleichzeitig kann wie gehabt ein Medientyp benannt werden:

```
@import "screen.css";
@import "print.css" print;
@import "mobile.css" handheld;
```

Anmerkung

Wird kein Medientyp angegeben, so ist die Reihenfolge der Importanweisungen für den Rang der importierten Regeln relevant – die Regeln der zuletzt importierten Datei besitzen Vorrang. Dasselbe ist der Fall, wenn mehrere Dateien in Zusammenhang mit dem gleichen Medientyp importiert werden.

Kompatibilität

Das `@import`-Rule wird durch *alle* aktuellen Browser unterstützt. Der inzwischen historische Netscape 4.x kennt die Regel hingegen nicht, weshalb sie sich für diesen Browser als Stylesheetweiche einsetzen lässt.

@media

Gültigkeit

CSS 2, CSS 2.1, CSS Mobile;

Beschreibung

Die Anweisung `@media` kann innerhalb eines Stylesheets als Schalter eingesetzt werden, um bestimmte, in einem Block zusammengefasste Stylesheetregeln medienabhängig auszuführen. Die Anweisung kann alternativ zu einem eigenständigen, medienspezifischen Stylesheet verwendet werden. Hierbei können jeweils ein oder mehrere Zielmedien (kommagetrennte Liste) angegeben werden, für die der Block gelten soll.

Beispielsweise kann ein mittels `id="nav"` gekennzeichneter Navigationscontainer für den Druck ausgeblendet werden:

```
/* Screenausgabe mit eingeblendeter Navigation: */

   #nav { display:block; }
```

```
/* Druckausgabe mit ausgeblendeter Navigation: */

@media print {
    #nav { display:none; }
    /* weitere Änderungen für Druckausgabe: */
    ...
}
```

Anmerkung

Die Reihenfolge der Anweisungen im Stylesheet ist dabei zu beachten – das obige Beispiel funktioniert nur, wenn die für den Druck einzusetzende Anweisung nach der allgemeinen Anweisung steht. Alternativ könnte man letztere ebenfalls in einen mit `@media` bedingten Block (für Bildschirmausgabe) setzen.

Kompatibilität

Das `@media`-Rule wird von allen aktuellen Browsern unterstützt. Mangelhaft oder gar nicht implementiert ist es lediglich in älteren oder historischen Browsern wie IE 4 und IE 5 (Win) sowie in Netscape 4.x.

@page

Gültigkeit

CSS 2, CSS 2.1

Beschreibung

Der Befehl `@page` betrifft die seitenorientierte Ausgabe des Dokuments, womit sowohl Druckausgabe als auch Projektion im Rahmen einer Präsentation abgedeckt werden soll. Bestimmt wird die Aufteilung des Dokuments in einzelne Seitenblöcke, wobei die Seitenabmessungen und -ränder bestimmt werden können.

Beispiel

```
@page {
    size: 21cm 29.7cm;
    margin-left: 3cm;
    margin-top: 4.5cm;
    margin-bottom: 3cm;
}
```

Kompatibilität

Derzeit wird die seitenorientierte Ausgabe mit `@page` in aktuellen Browsern nicht unterstützt.

19.3 CSS Selektoren CSS 1 und 2

Selektoren dienen dazu, eine CSS-Regel an ein Element oder eine Gruppe von Elementen zu binden. Sie stehen daher im Zentrum der Funktionalität von Stylesheets. Man unterscheidet einfache und zusammengesetzte Selektoren sowie Kontext- und Pseudoselektoren.

Universeller Selektor

Anwendbar auf

Alle Elemente

Syntax

```
*
```

Beschreibung

Der **universelle Selektor** (*wildcard*-Selektor) gilt für Elemente beliebigen Namens. Sein Einsatz macht hauptsächlich innerhalb von zusammengesetzten Selektoren oder Kontextselektoren Sinn.

Die folgende Regel ordnet für alle Elemente die Schriftart Arial an (alternativ könnte die Regel ebenso auf das `body`-Element angewendet werden):

```
* {font-family:arial;}
```

In Zusammenhang mit einem Attributselektor kann der universelle Selektor als Verdeutlichung gesetzt werden (darf jedoch auch entfallen):

```
*[alt] {border: dotted 1px black;}
```

… gilt für jedes Element, das ein `alt`-Attribut besitzt.

In zusammengesetzten Selektoren ist `*` ein Platzhalter für beliebige Elementbezeichner:

```
div * {color:red}
```

…gilt für jedes Element, das innerhalb eines `<div>`-Containers steht.

Folgende Regel gilt für alle `<div>`, die nicht *unmittelbar* innerhalb des durch einen Identifier gekennzeichneten Bereichs `#content` liegen. Es ist dabei irrelevant, was für ein Element (oder ob mehr als eines!) hierarchisch dazwischen liegen:

```
/* gilt, wenn ein Element zwischen #content und div steht: */

#content * div {...}
```

Zum Vergleich – die folgende Regel gilt ebenfalls, wenn der `<div>`-Container direkt in `#content` steht. Ohne Verwendung des universellen Selektors wäre eine einschränkende Regel wie die oben stehende nicht formulierbar:

```
/* gilt auch unmittelbar in #content: */

#content div {...}
```

Kompatibilität

Der universelle Selektor `*` wird von *allen* Browsern unterstützt. Ausnahmen sind die historischen Browser Netscape 4.x und Internet Explorer 4.0.

Typselektor

Anwendbar auf

Alle Elemente

Syntax

```
tagname
```

Beschreibung

Der Typselektor (Elementselektor) zielt auf Elemente des genannten **Tagnamens** (*generic identifier*). Er besteht daher im einfachsten Fall aus einem Tagbezeichner (sowohl ein HTML-Tagbezeichner als auch gegebenenfalls ein XML-Tagbezeichner).

Die folgende Stylesheetregel betrifft alle `<p>`-Container eines Dokuments:

```
p {font-family:arial; font-size:0.8em; }
```

Eine Regel mit Typselektor gilt global im gesamten Dokument für alle Elemente des übergebenen Typs.

Kompatibilität

Der Typselektor wird von *allen* Browsern unterstützt.

Gruppenselektor

Anwendbar auf

Alle Elemente

Syntax

```
selektorA, selektorB, selektorC
```

Beschreibung

Soll die gleiche Regel für mehrere Selektoren gelten, so können diese **gruppiert** werden. Hierfür werden die Einzelselektoren zu einer kommagetrennten Liste zusammengefügt.

Im einfachsten Fall ist es eine Liste aus Typselektoren. Hier wird für die Überschriften `<h1>`, `<h2>` und `<h3>` die gleiche Schrift angeordnet:

```
h1, h2, h3 {font-family:arial;}
```

Anmerkung

Die Gruppierung von Selektoren ist nicht auf Typselektoren beschränkt, sondern umfasst genauso zusammengesetzte Selektoren, Klassen, ID-Selektoren oder Selektoren mit Pseudoanteilen.

Beispiel

```
/* drei Klassenselektoren gruppiert: */
.klasseA, .klasseB, klasseC:hover {...}
```

```
/* zwei ID-Selektoren und ein Descendant-Selektor: */
#idA, #idB, div span {...}
```

Kompatibilität

Der Gruppenselektor wird von *allen* Browsern unterstützt.

Klassenselektor

Anwendbar auf

Alle Elemente mit `class`-Attribut

Syntax

```
.klassenname
```

Beschreibung

Definiert eine CSS-Klasse, die einem Element über das Attribut `class` zugewiesen werden kann.

Beispiel

```
.wichtig {border:solid 2px red;}
```

Kompatibilität

Der Klassenselektor wird von *allen* Browsern unterstützt.

Freie Klassen

Beschreibung

Eine Klasse ohne vorangestellten Typselektor wirkt auf jedes Element mit entsprechendem `class`-Attributwert. Einer solchen „freien" Klasse ist also quasi der universelle Selektor vorangestellt, der in diesem Zusammenhang jedoch entfallen kann.

Beispiel

```
*.wichtig
```

Beide folgenden Elemente werden gestylt:

```
<p class="wichtig">Dies hier ist wichtig!</p>
<div class="wichtig">Dies hier ist auch wichtig.</div>
```

Gebundene Klassen

Beschreibung

Wird dem Klassenselektor ein Typselektor vorangestellt, so wird die Wirkung der Klasse auf Elemente dieses Typs begrenzt.

```
p.wichtig {border:solid 2px red;}
```

Im folgenden Beispiel wird also nur der Inhalt des `<p>`-Containers, nicht aber der des `<div>`-Containers gestylt:

```
<p class="wichtig">Dies hier ist wichtig!</p>
<div class="wichtig">Dies hier wird nicht gestylt.</div>
```

Zuweisung mehrerer Klassen an einen Elementcontainer

Beschreibung

Einem Element können mehrere Klassen zugewiesen werden, die gleichzeitig angewendet werden. Hierfür wird im `class`-Attribut eine durch Leerzeichen getrennte **Liste** mit Klassenbezeichnern angegeben

Beispiel

```
<p class="hinweis wichtig">Dies ist ein wichtiger Hinweis.</p>
```

Bei Konflikten zwischen den Properties der anzuwendenden Klassen zählt nicht die Reihenfolge der Nennung im `class`-Attribut, sondern die ihrer Deklaration im Stylesheet – Vorrang hat das zuletzt definierte Property.

Anmerkung

Aus der Möglichkeit, Listen zu übergeben, folgt eine mögliche Alternativschreibweise für den Klassenselektor (hier eine freie Klasse):

```
*[class~=klassenname]
```

Bei der Umschreibung durch einen Attributselektor müsste man den Operator ~= (d.i. „enthält“) verwenden, da sonst nur diejenigen Elemente erfasst würden, die ausschließlich den übergebenen Klassennamen besitzen.

Wahl des Bezeichners für Stylesheetklassen

Klassennamen

Die im Prinzip frei wählbaren Klassennamen dürfen beliebig aus **Buchstaben** (`a-z`, `A-Z`), **Ziffern** (`0-9`) und dem Bindestrich (`-`) bestehen. Sie müssen mit einem Buchstaben beginnen, Sonderzeichen sind verboten.

Anmerkung

Auch der Unterstrich _ ist zwar inzwischen als Teil eines Bezeichners durch die CSS-Spezifikation erlaubt, Klassenbezeichner mit Unterstrich werden jedoch in älteren Browsern nicht allgemein unterstützt und sollten vermieden werden.

ID-Selektor

Anwendbar auf

Alle Elemente mit `id`-Attribut

Syntax

```
#IDbezeichner
```

Beschreibung

Der ID-Selektor wirkt auf *genau ein* Element, das ein `id`-Attribut mit dem übergebenen Bezeichner besitzt. Er wird in der Regel verwendet, um gezielt Dokumentbereiche oder logische Elemente zu stylen, die sich im Dokument nicht wiederholen (andernfalls greift man auf Klassen zurück).

Anmerkung

Es ist zur Verdeutlichung zulässig, den Elementnamen des Containers, der den ID trägt, mit in den Selektor zu nehmen

Beispiel

```
p#einleitung { }
```

erfasst einen `<p>`-Container, der den `id="einleitung"` besitzt. Da der Identifier eindeutig ist, ist dies prinzipiell nicht erforderlich, kann aber die Lesbarkeit des Stylesheets erhöhen.

Kompatibilität

Der ID-Selektor wird von *allen* Browsern unterstützt.

Descendant-Selektoren (CSS 1)

Anwendbar auf

Alle Elemente

Syntax

```
vorfahre abkoemmling
```

Beschreibung

Dieser zusammengesetzte Selektor erfasst alle Elemente vom Typ `abkoemmling`, die (irgendwo oberhalb in der Hierarchie) einen Vorfahren vom Typ `vorfahre` besitzen. Die Einzelselektoren werden durch mindestens ein Leerzeichen getrennt.

Beispiel

```
a b
```

erfasst `<b>` in `<a>`, wenn folgende Struktur vorliegt

```
<a>
 <b> ... </b>
</a>
```

aber auch, wenn beispielsweise gilt

```
<a>
 <c>
  <b> ... </b>
 </c>
<a>
```

Die hierarchische Lage des `b`-Knotens innerhalb von `a` ist unerheblich – die Regel gilt also für Kindknoten genauso, wie für beliebig tief gestaffelte Abkömmlinge (Kindknoten sind eine Untermenge der Abkömmlinge).

Kompatibilität

Der Descendant-Selektor wird von *allen* Browsern mit Ausnahme des historischen Netscape 4.x unterstützt.

Child-Selektor (CSS 2)

Anwendbar auf

Alle Elemente

Syntax

```
elternelement > kindelement
```

Beschreibung

Dieser zusammengesetzte Selektor erfasst alle Elemente vom Typ `kindelement`, die ein Elternelement vom Typ `elternelement` besitzen. Zwischen den Einzelselektoren steht das Kombinatorsymbol `>`, das von Leerzeichen umgeben sein darf (die Leerzeichen dürfen entfallen, da das Kombinatorzeichen nicht Teil eines Elementnamens sein kann).

Beispiel

```
a > b
```

erfasst `<b>` in `<a>`, wenn folgende Struktur vorliegt

```
<a>
 <b> ... </b>
</a>
```

nicht jedoch, wenn beispielsweise gilt

```
<a>
 <c>
  <b> ... </b>
 </c>
<a>
```

... da der Childselektor nur für die unmittelbar in `<a>` enthaltenen `<b>` gilt.

Kompatibilität

Der Child-Selektor wird von allen modernen Browsern unterstützt (Mozilla, Firefox, Safari, Opera, auch IE5/Mac). Der Internet Explorer für Windows hingegen kennt den Child-Selektor erst ab Version 7.0.

Adjacent-Sibling-Selektor (CSS 2)

Anwendbar auf

Alle Elemente

Syntax

```
vorgaenger + nachfolger
```

Beschreibung

Dieser zusammengesetzte Selektor erfasst ein Element `nachfolger`, wenn ihm ein Element `vorgaenger` *direkt vorangeht* (hierunter versteht man, dass der andere Container des Vorgängers unmittelbar vor dem Starttag des Nachfolgers *geschlossen* wird). Zwischen den Einzelselektoren steht das Kombinatorsymbol `+`, optional umgeben von Leerzeichen.

Beispiel

```
a + b
```

erfasst `<b>` nach `<a>`, wenn folgende Struktur vorliegt:

```
<a> ... </a>
<b> ... </b>
```

nicht jedoch, wenn beispielsweise gilt (`<b>` steht in `<a>`):

```
<a>
  <b> ... </b>
<a>
```

und ebenfalls nicht, wenn gilt (`<c>` zwischen `<a>` und `<b>`):

```
<a> ... </a>
<c> ... </c>
<b> ... </b>
```

... im zweiten Fall ist `<b>` Kindelement und nicht Nachfolger von `<a>`, im dritten Fall ist `<b>` nicht *unmittelbarer* Nachfolger von `<a>`, sondern von `<c>`.

Kompatibilität

Der Adjacent-Sibling-Selektor wird von allen modernen Browsern unterstützt (Mozilla, Firefox, Safari, Opera, auch IE5/Mac). Der Internet Explorer für Windows hingegen kennt den Child-Selektor erst ab Version 7.0.

19.3.1 Attributabhängige Selektoren

Auch die Attributselektoren wählen Elementcontainer innerhalb des Dokuments aus, prüfen dabei jedoch deren **Attributkontext**.

Gewählt werden Elemente, die entweder ein bestimmtes Attribut besitzen, ohne dass dessen Wert eine Rolle spielt, Elemente, deren genanntes Attribut gleichzeitig einen bestimmten Wert besitzt oder (im Rahmen einer Tokenliste) den entsprechenden Vergleichswert enthält.

Vorhandensein eines Attributs

Anwendbar auf

Alle Elemente mit entsprechendem Attribut

Syntax

`[attributname]`, eigentlich Abkürzung für `*[attributname]`

Beschreibung

Ein freier Attributselektor gilt für alle Elemente, für die ein Attribut mit entsprechendem Namen *explizit* gesetzt ist (d.h. im Quelltext vorhanden ist).

Die Wirkung kann auf Elemente mit bestimmtem Namen beschränkt werden.

tagname`[attributname]`

Dieser Selektor erfasst alle Elemente vom Typ `tagname`, die über ein Attribut `attributname` verfügen.

Beispiel

Ein Link mit Accesskey kann hervorgehoben werden:

```
a[accesskey] { background-color:#ff9999; }
```

Attribut mit bestimmtem Wert

Syntax

```
[attributname=attributwert]
```

Beschreibung

Dies gilt für alle Elemente, die ein Attribut `attributname` besitzen, wenn dieses den Wert `attributwert` enthält. Der Wert wird als Zeichenkette mit Begrenzern übergeben und darf daher Leerzeichen enthalten.

Beispiel

```
img[alt=""] {border:solid 2px red;}
```

… hebt Bilder mit leerem `alt`-Attribut hervor.

Attribut mit Wert innerhalb einer Wertliste

Syntax

```
[att~=teilwert]
```

Beschreibung

Ebenso können Elemente erfasst werden, die Attribute in Form eines Listentyps (z. B. `IDREFS`, `NMTOKENS`) besitzen, sofern ein Teilwert des Attributwertes dem hier übergebenen Wert entspricht. Der übergebene Wert darf keine Leerzeichen enthalten.

Anmerkung

Als Attribute vom Listentyp sind definiert:

Das `rel`- und `rev`-Attribut (von `<a>` und `<link>`), das `headers`-Attribut von `<td>` sowie das Universalattribut `class`.

Attributselektoren für Sprachcodes

Syntax

```
[att|=val]
```

Beschreibung

Speziell für Elemente mit zugewiesenem Sprachcode ist diese Form des Attributselektors geeignet. Hierbei muss der Attributwert mit dem übergebenen Wert beginnen, wobei unmittelbar ein Bindestrich folgen darf.

Beispiel

```
<p lang="EN-us"> ... </p>
```

wird erfasst durch

```
[att|="EN"]
```

Kompatibilität

Der Attribut-Selektor wird von allen modernen Browsern unterstützt (Mozilla, Firefox, Safari, Opera). Der Internet Explorer für Windows hingegen kennt den Attribut-Selektor erst ab Version 7.0. Der IE 5 für MacOS und der historische Netscape 4.x unterstützen diesen Selektor nicht.

19.3.2 Pseudoelemente

CSS-Pseudoelemente dienen zum Stylen von **Bereichen** innerhalb eines Elements oder zum Stylen von **generiertem Content**, zielen also im Gegensatz zu einem Typselektor oder anderen »normalen« Selektoren nicht auf ein real existierendes Element.

:after, :before

Gültig in
CSS 2, CSS 2.1
Anwendbar auf
Alle Elemente
Syntax
`selektor:after`
`selektor:before`
Beschreibung

Die Pseudoelemente `:after` und `:before` dienen zum Einfügen von beliebigem Inhalt *nach* bzw. *vor* dem eigentlichen Inhalt des über den Selektor gewählten Elements. Der einzufügende Inhalt wird mit dem Property `content` angegeben und kann mit weiteren Properties beliebig gestylt werden.

Eingefügt werden kann beliebiger Text (in Anführungszeichen zu übergeben) oder eine Grafik (mit dem Befehl `url()`), aber auch generierter Content wie Anführungszeichen oder Zähler (entsprechende Keywords ohne Anführungszeichen).

Beispiel

(auf `<div>` mit entsprechenden `id`-Werten angewendet)

```
div#eins:before { content:url('pfeil.gif'); }
div#zwei:after { content:url('pfeil.gif'); }
div#drei:before { content:'Blindtext: '; font-weight:bold;}
div#vier:before { content: open-quote; }
div#vier:after { content: close-quote; }
```

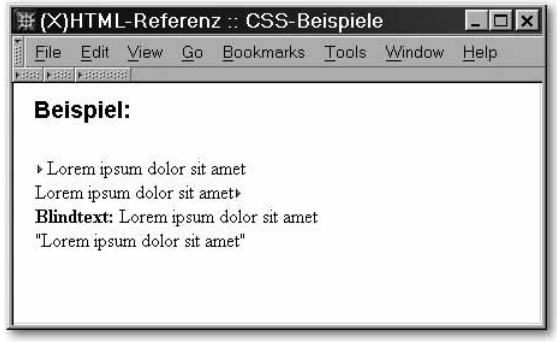

Abbildung 19.4 Grafiken und Text mit :before und :after einfügen

:first-letter

Gültig in
CSS1, CSS 2, CSS 2.1
Anwendbar auf
Alle Elemente
Beschreibung
Das Pseudoelement `:first-letter` stylt das erste Zeichen des Inhalts des Elements, das über den Selektor gewählt wurde. Es kann beispielsweise zur Erzeugung eines Initials verwendet werden.
Beispiel

```
p        { font-family:arial, sans-serif;
           font-size:0.8em;}
p:first-letter { font-size:3em;
           font-weight:bold; float:left; }
```

Abbildung 19.5 Wirkung von :first-letter

:first-line

Gültig in
CSS1, CSS 2, CSS 2.1
Anwendbar auf
Blockelemente
Beschreibung
Das Pseudoelement `:first-line` stylt die erste Zeile des Inhalts des mittels des Selektors gewählten Elements. Der Inhalt (bzw. die Länge) der Zeile hängt vom verfügbaren Platz ab, ist also unter Umständen veränderlich.
Beispiel

```
p        {font-family:arial, sans-serif;
         font-size:0.8em;}
p:first-line {font-weight:bold;}
```

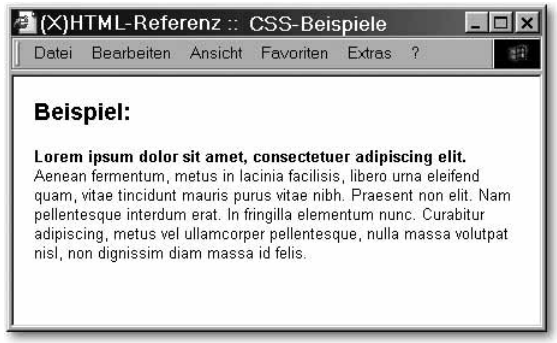

Abbildung 19.6 Wirkung von :first-line

19.3.3 Pseudoklassen

Die so genannten **Pseudoklassen** werden in Zusammenhang mit dem Stylen von Links benötigt (theoretisch sind einige davon jedoch auf alle Elemente anwendbar) – sie beziehen sich im Gegensatz zu »echten« Klassen nicht auf das mit dem vorausgehenden Selektor bezeichnete Element, sondern dessen *Zustand* und werden daher (mit Ausnahme von `:lang`) auch oft auch als *dynamische Klassen* bezeichnet. Pseudoklassen werden allgemein folgendermaßen zugewiesen:

```
selektor:pseudoklasse
```

Browserkompatibilität

Die dynamischen Pseudoklassen sind im Internet Explorer und Opera (bis 6.0) nur auf das `<a>`-Element anwendbar, in Mozilla und anderen Browsern, die die gleiche Rendering-Engine verwenden, auch auf beliebige andere Elemente.

Pseudoklassen und Links:

Beim Zuweisen mehrerer Pseudoklassen an Links ist die **Definitionsreihenfolge** maßgeblich. Folgende Reihenfolge der Deklarationen sollte im Stylesheet unbedingt eingehalten werden:

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

Will man gleichzeitig auch den **Fokus-Zustand** `:focus` erfassen, so muss dieser vor dem `:active`-Zustand definiert werden:

- `a:link`

- a:visited
- a:hover
- **a:focus**
- a:active

:active

Gültig in
CSS1, CSS 2, CSS 2.1
Anwendbar auf
Alle aktivierbaren Elemente
Beschreibung
Die Pseudoklasse :active stylt den Zustand aktiver Elemente, also solche, die durch den User angeklickt werden (onmousedown). Das Element bleibt für die Dauer des Klicks aktiv. Die Pseudoklasse wird zwar meist für Links verwendet, ist jedoch für alle anklickbaren (aktivierbaren) Elemente gestattet. Dies ermöglicht es (in der Theorie), auch Formularfeldern, Tabellenzellen oder -Bereichen aktive Zustände zuzuweisen.

:focus

Gültig in
CSS 2, CSS 2.1
Anwendbar auf
Alle aktivierbaren Elemente, Formulareingabeelemente
Beschreibung
Die Pseudoklasse :focus stylt den Fokus-Zustand von Elementen. Dies trifft auf Hyperlinks zu, die per Tabulator angewählt, aber noch nicht aktiviert sind, sowie auf Formulareingabefelder, die angeklickt wurden. Es kann jeweils nur ein einziges Element den Fokus besitzen.
Anmerkung
Internet Explorer kann :focus nicht auf Formularelemente anwenden und unterscheidet bei Links nicht zwischen :focus und :active. In anderen Browsern kann ein per Tab angewählter Link (Fokuszustand) dagegen gezielt gestylt werden.
Beispiel
`input:focus { background-color:#DDDDDD; }`
Ein per Tab oder Klick angewähltes Formularfeld erhält eine hellgraue Hintergrundfarbe.

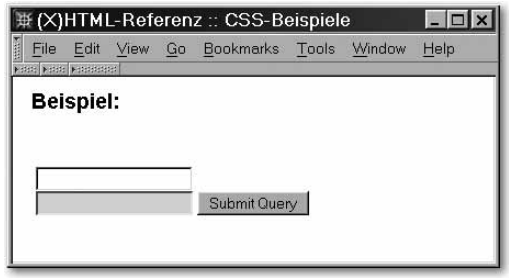

Abbildung 19.7 Wirkung von :focus auf Formulareingabefelder (Mozilla)

:hover

Gültig in
CSS 2, CSS 2.1
Anwendbar auf
Alle Elemente
Beschreibung
Die dynamische Pseudo-Klasse `:hover` bestimmt das Aussehen des Links (bzw. des gestylten Elements) im Zustand des MouseOvers. Prinzipiell kann `:hover` zwar auf alle Elemente angewendet werden, im Internet Explorer und älteren Opera-Browsern beschränkt sich der Einsatz jedoch auf Hyperlinks.
Beispiel

Beliebt ist das Aktivieren eines Unterstrichs für Links. Dieser muss für den Grundzustand zunächst deaktiviert werden:
`a:link {text-decoration:none;}`
`a:hover {text-decoration:underline;}`
Eine weitere interessante Verwendungsmöglichkeit besteht in Hover-Zuständen für Absätze, die dem Nutzer das Lesen erleichtern, wenn er dem Text mit der Maus folgt:
`p:hover {background-color:#dddddd;}`

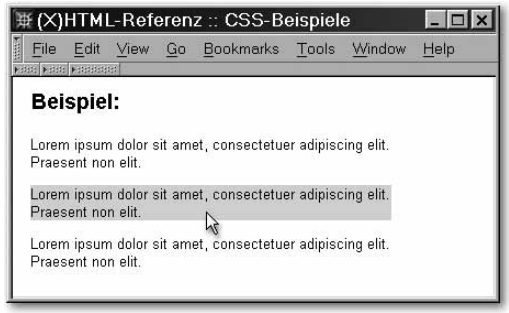

Abbildung 19.8 Die Pseudoklasse :hover auf Absätze angewendet (Mozilla)

:lang

Gültig in
CSS 2, CSS 2.1
Anwendbar auf
Alle Elemente
Beschreibung
Die Pseudoklasse `:lang` formatiert ein Element in Abhängigkeit der eingestellten Sprache. Diese muss durch das `lang`-Attribut des Elements angegeben sein.
Beispiel

```
p       { color:#999999; font-family:arial;}
p:lang(de)  { color:#000000; }
p:lang(en)  { color:#666666; font-weight:bold;}
```

Die Ausgangstextfarbe eines Absatzes ist hellgrau. Inhalte explizit deutschsprachiger Absätze werden schwarz dargestellt, als englischsprachig markierte Absätze erhalten dagegen grau als Textfarbe, ihr Inhalt wird fett ausgezeichnet.

Das `lang`-Attribut muss im Quelltext mit entsprechendem Wert gesetzt sein:

```
<p>Hier besteht keine Sprachangabe per lang-Attribut.</p>
<p lang="de">Dies ist deutscher Text, gekennzeichnet durch lang="de". </p>
<p lang="en">This is written in english, marked with lang="en".</p>
```

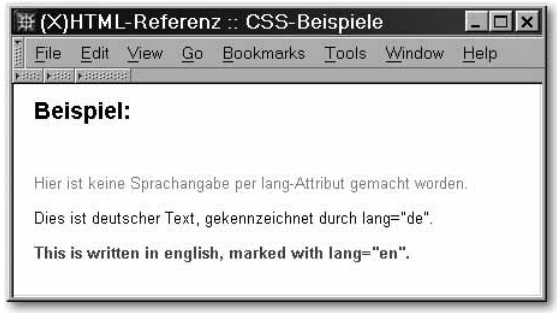

Abbildung 19.9 Die Wirkung der :lang-Pseudoklasse

:link

Gültig in
CSS1, CSS 2, CSS 2.1
Anwendbar auf
Hyperlinks
Beschreibung
Die Pseudo-Klasse `:link` bestimmt die Darstellung nicht besuchter Hyperlinks, also die Ausgangspräsentation des Links.

Beispiel
Der Linkunterstrich wird für unbesuchte Links entfernt
`a:link {text-decoration:none;}`

:visited

Gültig in
CSS1, CSS 2, CSS 2.1
Anwendbar auf
Hyperlinks
Beschreibung
Die Pseudo-Klasse `:visited` bestimmt die Darstellung besuchter Hyperlinks. Auf diese Weise können besuchte von unbesuchten Links unterschieden werden, um dem Nutzer der Seite den Überblick zu erleichtern.
Beispiel
Besuchte Links werden ausgegraut:
`a:visited {color:#666666;}`

19.3.4 Pseudo-Kontextselektoren

Auch in Zusammenhang mit Kontextselektoren spricht man von einem »Pseudoselektor«, sofern dieser einen künstlich beschränkten Bereich erfasst.

:first-child

Gültig in
CSS 2, CSS 2.1
Anwendbar auf
Alle Elemente
Beschreibung
Spricht innerhalb der Dokumentenstruktur ein Element an, wenn es das erste Kind seines Elternelements ist. In einer `<ul>`-Liste ist der erste `<li>`-Eintrag das erste Kind von `<ul>` und kann mit `li:first-child` anders formatiert werden.
Anmerkung
Nicht das Elternelement ist hier ausschlaggebend, sondern lediglich, dass das Zielelement ein »erstes Kindelement« ist, egal von welchem Elternelement.

Beispiel

```
li:first-child { font-weight:bold; }
```

ist gleichbedeutend mit

```
* li:first-child { font-weight:bold; }
```

erfasst gleichermaßen das erste `<li>`-Element von `<ol>`- wie `<ul>`-Listen. Um die Wirkung beispielsweise auf `<ul>`-Listen zu beschränken, schreiben Sie einfach unter Zuhilfenahme des Descendant-Selektors:

```
ul li:first-child { font-weight:bold; }
```

Abbildung 19.10 Erfassen eines ersten Listenelements mit :first-child

Analog können Sie die erste Zeile oder Spalte einer Tabelle erfassen:

```
tr:first-child { background-color:#dddddd; }
```

```
td:first-child { background-color:#dddddd; }
```

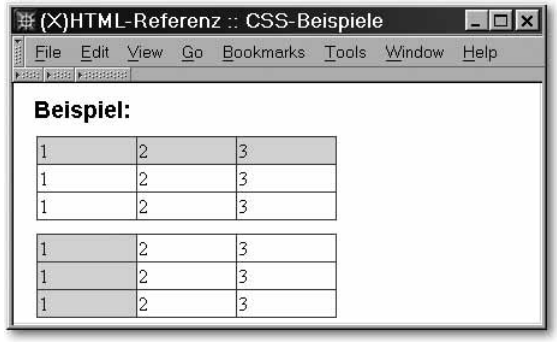

Abbildung 19.11 Erfassen der ersten Tabellenzeile oder -spalte mit :first-child

19.4 Werte und Maßeinheiten

Längenangaben

Abmessungen können in CSS in Form von Längenwerten mit Maßeinheit bestimmt werden. Entgegen der Gepflogenheit in HTML sind keine Zahlenwerte ohne Maßeinheit, die als Pixelwerte interpretiert werden, gestattet, sondern es ist stets eine Einheit zu benennen.

Man unterscheidet zwischen *absoluten* und *relativen* Längenwerten. Die ersteren bestimmen eine exakte Darstellungsgröße, die an genormte Einheiten gebunden ist, die letzteren eine Darstellungsgröße in Abgängigkeit von äußeren Bedingungen wie die festgelegte Basisgröße eines Ausgabemediums oder eines Elements, auf die sich die relative Angabe bezieht.

Absolute Größenangaben sind demzufolge geeignet für Ausgabemedien, deren Randbedingungen vorhersehbar sind, beispielsweise den Ausdruck von Dokumenten, bei denen wegen festgelegter Papiermaße entsprechend exakte Angaben sinnvoll sind (Medientyp `print`).

Zu den absoluten Maßeinheiten in diesem Sinne zählen die Einheiten:

`cm` (Zentimeter), `mm` (Millimeter), `in` (Zoll), `pt` (Punkt), `pc` (Pica)

Relative Größenangaben sind hingegen geeignet für Ausgabemedien mit wechselnden Randbedingungen, also beispielsweise Bildschirmausgabe, wo neben den Vorlieben des Nutzers als Parameter die Abmessungen und Auflösung des Bildschirmes oder die Größe des Browserfensters berücksichtigt werden müssen.

Zu den relativen Maßeinheiten in diesem Sinne zählen:

`em` (M-Breite), `ex` (x-Höhe), `px` (Pixel), `[zahl]%` (Prozentwert)

em – Maßeinheit

Die Bezeichnung **em** rührt daher, dass sich die Einheit auf die Breite des Großbuchstabens »M« im augenblicklich gewählten Schrifttyp bezieht. Die Einheit entspricht daher der Breite des amerikanischen Gedankenstrichs »em-dash«, typographisch also einer Geviertbreite. Bei einer 10 Punkt Schrift ist 1 em also 10 Punkt breit

Zur Definition von Schriftgrößen, Zeilenhöhen und – wie noch gezeigt werden wird – auch beliebigen weiteren Aufgaben, bei denen Maßangaben benötigt werden, ist die relative Größenangabe in *em* eine höchst interessante und flexible Alternative, jedoch keine ganz pflegeleichte: Da sich die Einheit stets auf einen *Basiswert* bezieht, dieser jedoch je nach Bezugspunkt wechseln kann, ist eine Bemessung in em auch für manche Überraschung gut.

Grundsätzlich lässt sich sagen, dass der Basiswert für den Browser mit der vom Nutzer dort eingestellten Schriftgröße identisch ist. Der Schriftgrad »mittel« im Internet Explorer bzw. »100 % (Original Size)« entspricht dabei einer Schrifthöhe von 16 px.

Aber Vorsicht – dies muss nicht so sein: Hat der Nutzer im Browser eine andere Schriftgröße eingestellt, weil er vielleicht mit sehr großer Bildschirmauflösung arbeitet, so ist der Basiswert (in Pixeln ausgedrückt) wiederum ein anderer.

In `em` ausgedrückt ist der Basiswert des `<body>`-Containers von Haus aus jedoch stets `1em`, was man lax ausgedrückt auch mit »100 % Basisschriftgröße« übersetzen kann. Der Wert bezieht sich auf die gesamte Seite und gilt für den `<body>`-Container und alle seine Inhalte als Ausgangswert.

Selbstverständlich ist der Basiswert nicht festgeschrieben, sondern kann per CSS-Styleanweisung auf einen beliebigen anderen Wert gesetzt werden. Dies kann für den `<body>` geschehen, aber auch für jeden anderen Container. Beispielsweise kann eine Überschrift auf `1.5em` gesetzt werden, was »anderthalbfache Basisschriftgröße« bedeutet. Gehen wir von unveränderter Basisschriftgröße aus (d. h. für den Body existiert keine explizite Anweisung), so können wir annehmen, dass die bewusste Überschrift mit ca. 24 Pixel Schrifthöhe angezeigt wird. Wird jedoch der Basisschriftgrad für den Body auf beispielsweise `2em` gesetzt, so dient dieser Wert nun als Basis für die Überschriftenformatierung – die Überschrift wird nun doppelt so groß (bleibt allerdings im gleichen relativen Größenverhältnis).

Lehrsatz

Eine `em`-Angabe bezieht sich stets auf die Basisgröße des übergeordneten Containers.

Bei größerem Verschachtelungsgrad oder unreflektierter bzw. zweckwidriger Verwendung von Styledefinitionen, kann die oben stehende, einfache Regel durchaus zu unübersichtlichen Ergebnissen führen, da sich die Basiswerte multiplizieren. Ist für `<body>` eine Basis von `0.8em` angeordnet und für `<p>` wiederum eine von `0.8em`, so liegt die summierte Basisschriftgröße innerhalb von `<p>` bei `0.64em`. Diese dient dann als Basis für weiter innen liegende Container.

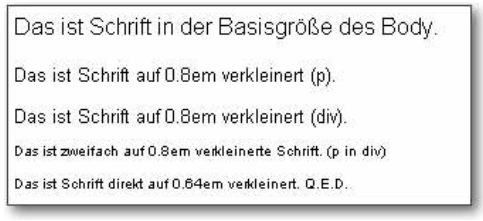

Abbildung 19.12 Wirkung verschachtelter em-Bemaßungen

em	px	~px	em	px	~px	em	px	~px
0.5em	**8.0**	**8**	**1.5em**	**24.0**	**24**	**2.5em**	**40.0**	**40**
0.55em	8.8	9	1.55em	24.8	25	2.55em	40.8	41
0.6em	9.6	10	1.6em	25.6	26	2.6em	41.6	42
0.65em	10.4	10	1.65em	26.4	26	2.65em	42.4	42
0.7em	11.2	11	1.7em	27.2	27	2.7em	43.2	43
0.75em	**12.0**	**13**	**1.75em**	**28.0**	**28**	**2.75em**	**44.0**	**44**
0.8em	12.8	13	1.8em	28.8	29	2.8em	44.8	45
0.85em	13.6	14	1.85em	29.6	30	2.85em	45.6	46
0.9em	14.4	15	1.9em	30.4	31	2.9em	46.4	46
0.95em	15.2	15	1.95em	31.2	32	2.95em	47.2	47
1em	**16.0**	**16**	**2em**	**32.0**	**32**	**3em**	**48.0**	**48**
1.05em	16.8	17	2.05em	32.8	33	3.05em	48.8	49
1.1em	17.6	18	2.1em	33.6	34	3.1em	49.6	50
1.15em	18.4	18	2.15em	34.4	34	3.15em	50.4	50
1.2em	19.2	19	2.2em	35.2	35	3.2em	51.2	51
1.25em	**20.0**	**20**	**2.25em**	**36.0**	**36**	**3.25em**	**52.0**	**52**
1.3em	20.8	21	2.3em	36.8	37	3.3em	52.8	53
1.35em	21.6	22	2.35em	37.6	38	3.35em	53.6	54
1.4em	22.4	22	2.4em	38.4	38	3.4em	54.4	54
1.45em	23.2	23	2.45em	39.2	39	3.45em	55.2	55
1.5em	**24.0**	**24**	**2.5em**	**40.0**	**40**	**3.5em**	**56.0**	**56**

Tabelle 19.3 Schriftgröße in em, theoretische und dargestellte Pixelhöhe (Basisgröße 1 em)

Schriftgrößen unter `0.5em` sind bei einer Basisgröße von `1em` nicht lesbar. Selbstverständlich sind für die Pixelhöhe nur ganzzahlige Werte realisierbar, wobei sich die Maße auf die so genannte **Kegelgröße** der Schrift beziehen. Die Abmessungen (Versalhöhe plus Unterlänge) der dargestellten Zeichen sind also meist (geringfügig) kleiner[3] als die laut der oben stehenden Tabelle zu erwartenden.

Selbstverständlich ist eine Schrifthöhe in Bruchteilen von Pixeln, wie sie sich rechnerisch ergibt, für den Browser nicht umsetzbar – er wird daher stets zu einem ganzzahligen, benachbarten Wert runden. Hierbei ist zu beachten, dass verschiedene `em`-Werte auch bei gleicher gerundeter Pixelhöhe unterschiedliche Schriftbilder erzeugen, da außer der Schrifthöhe auch andere Parameter beeinflusst werden (Fettegrad, Mittellänge und Dickte der Zeichen[4]).

3 In der Regel gilt: Kegelgröße = (Versalhöhe + Unterlänge) x 1.07.

4 Falls Sie sich an dieser Stelle für Typographie zu interessieren beginnen, möchte ich Ihnen *Mut zur Typographie* von Gulbins/Kahrmann, Springer Verlag Heidelberg, empfehlen.

ex – Maßeinheit

Weniger gebräuchlich als `em` ist die Einheit **ex** für Schriftgrößenangaben. Es handelt sich ebenso wie `em` um eine relative Angabe, die sich auf eine Basisgröße bezieht – in diesem Falle die so genannte **x-Höhe** der Schriftgröße des umliegenden Containers. Gemeint ist in der Tat die Höhe des Kleinbuchstabens x, also die **Mittellänge** des Schriftschnitts.

Diese Höhe beträgt (für Arial) ca. 0,556 der Schrifthöhe (Versalhöhe plus Unterlänge). Für andere Schriften ist dieses Verhältnis allerdings ein anderes, so dass die Wirkung einer `ex`-Angabe stark von der Wahl der Schrift abhängt.

Im Gegensatz zur Angabe `1em` ist die Angabe `1ex` daher nicht neutral, sondern führt zu einer Reduzierung der Schriftgröße – der neutrale Wert betrüge hier ca. 1.8 ex.

Prozentangaben

Prozentangaben können sowohl positiv als auch negativ sein, also ein Vorzeichen erhalten. Das Prozentzeichen muss ohne dazwischenstehenden Leerraum an die numerische Angabe angehängt werden.

Jede Prozentangabe ist stets *relativ*, beruht also auf einer Bezugsgröße, die entweder aus dem übergeordneten Elementcontainer abgeleitet wird (der Regelfall) oder aus einer anderen Eigenschaft desselben Elementcontainers.

Beispielsweise richtet sich eine Prozentangabe für die Zeilenhöhe `line-height` nach der Schriftgröße `font-size`, wie sie innerhalb des Elementcontainers gilt. Eine Prozentangabe für Höhe oder Breite eines Containers hingegen wird an der Höhe oder Breite des umgebenden Containers (des Elternelements) gemessen.

[zB] *Beispiel:*

```
p {width: 70%; }

p.eng { line-height: -10%; }
```

Anwendung: Prozentangaben können bei folgenden CSS-Properties eingesetzt werden:

`background-position`, `font-size`, `height`, `left`, `line-height`, `margin`, `max-height`, `max-width`, `min-height`, `min-width`, `padding`, `right`, `text-indent`, `top`, `vertical-align`, `width`

Anmerkung: Im Falle der Vererbung eines durch eine Prozentangabe ermittelten Wertes an einen untergeordneten Containers wird der tatsächliche Wert (z. B. in Pixel) vererbt, nicht die Prozentangabe selbst.

Farben

In CSS können Farben verschiedenen Elementeigenschaften zugewiesen werden, wobei man zwischen drei grundsätzlichen Methoden der Farbzuweisung und zwei von diesen abgeleiteten Varianten unterscheiden kann.

- **Farbname**
  Es kann einer der vordefinierten Farbnamen verwendet werden. In CSS 2.1 sind *siebzehn Farbnamen* als Grundfarben festgelegt und weitere 130 Farbbezeichner für die restlichen gängigen Farbtöne. Diese entsprechen größtenteils den durch Netscape eingeführten Farbnamen, die aber inzwischen auch in SVG 1.0 und als so genannte »X11"-Farben spezifiziert sind. Da die Unterstützung durch Browser nicht in allen Einzelfällen gewährleistet ist, sind Angaben in Hexadezimal- oder RGB-Form vorzuziehen.

  Beispiel: **[zB]**

```
p { background-color: red }
```

- **Hexadezimalfarbangabe**
  Die Farbangabe wird in die drei Grundfarbanteile (Rot, Grün, Blau) der additiven Farbsynthese aufgeschlüsselt als Hexadezimalzahlpärchen RRGGBB vorgenommen, wobei der Angabe das Hash-Symbol # vorangestellt wird.

  Beispiel: **[zB]**

```
p { background-color: #aaccff }
```

- **Hexadezimalfarbangabe, gekürzt**
  Sind jeweils beide Hexadezimalzahlen einer Farbkomponente identisch, so ist das Fortlassen des jeweils zweiten Zeichens als Verkürzung gestattet. Statt (abstrakt) `#RRGGBB` kann also `#RGB` geschrieben werden.

  Beispiel: **[zB]**

```
p { background-color: #acf }     /* entspricht: #aaccff */
```

- **RGB-Wert, numerisch**
  Die RGB-Werte können anstatt in hexadezimaler auch direkt in dezimaler Form übergeben werden, wobei sie mit dem Ausdruck `rgb( )` zu klammern sind. Für jede Farbkomponente sind Werte zwischen 0 (Minimalwert) und 255 (Maximalwert) möglich. Die Angaben für die Einzelkomponenten müssen durch Komma getrennt werden.

[zB] *Beispiel:*

```
p { background-color: rgb(255, 0, 0) }    /* rot */
```

- **RGB-Wert, prozentual**
  Alternativ zu einer numerischen Angabe können die Anteile der Farbkomponenten auch als Prozentangabe übergeben werden. Der Prozentwert bezieht sich dabei auf den Maximalwert 255.

[zB] *Beispiel:*

```
p { background-color: rgb(100%, 0%, 0%) }  /* rot */
```

19.5 Properties CSS 1 und 2

19.5.1 Textproperties

direction

CSS-Level	CSS 2
Anwendung	global
Vererbung	ja
Werte	*rtl, ltr*
Defaultwert	*ltr*

Das `direction`-Property bestimmt die Schreibrichtung innerhalb des erfassten Elements. Texte können somit auch rechts nach links dargestellt werden, wodurch es unter anderem möglich ist, Texte in Hebräisch oder Arabisch darzustellen, bei denen die umgekehrte Leserichtung gilt.

- `direction: ltr` (Default)
  Textrichtung links nach rechts
- `direction: rtl`
  Textrichtung rechts nach links

[zB] *Beispiel:*

```
p.hebr { direction:rtl; }

p.engl { direction:lrt; }
```

Kompatibilität
Die `direction`-Property wird derzeit von *keinem* Browser unterstützt.

letter-spacing

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*normal, inherit, [laenge]*
Defaultwert	*normal*

Das `letter-spacing`-Property bestimmt den Abstand zwischen den Zeichen eines Textes. Der Defaultwert ist `"normal"`, was den für den jeweiligen Schrifttyp üblichen Wert bedeutet. Eine Längenangabe definiert alternativ den gewünschten Zeichenabstand, wobei ein positiver Wert den Abstand vergrößert, ein negativer diesen verringert.

Beispiel: **[zB]**

```
blockquote { letter-spacing: 0.1em }
```

Kompatibilität

Die `letter-spacing`-Property wird von *allen* aktuellen Browsern unterstützt. Lediglich der historische Netscape 4.x kennt diese Property nicht.

line-height

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*normal, inherit, [faktor], [laenge], [prozent]*
Defaultwert	*normal*

Das `line-height`-Property bestimmt die Zeilenhöhe innerhalb eines Containers und damit indirekt den Zeilenabstand. Wird eine Zeilenhöhe niedriger als die vorherrschende Schrifthöhe angegeben, so führt dies zu einer Überlagerung der Zeilen untereinander.

Als Werte können Multiplikationsfaktoren, Längenmaße mit Einheit oder Prozentangaben übergeben werden, die sich auf den Grundlinienabstand der Zeilen in Abwesenheit des Properties bezieht.

Die Zeilenhöhe entspricht per Default dem 1.2-fachen der Schrifthöhe.

[zB] *Beispiel:*

```
p { line-height: 0.9;}    /* Faktor */
p { line-height: 1.2em;}  /* Längenangabe */
p { line-height: 140%;}   /* Prozentangabe */
```

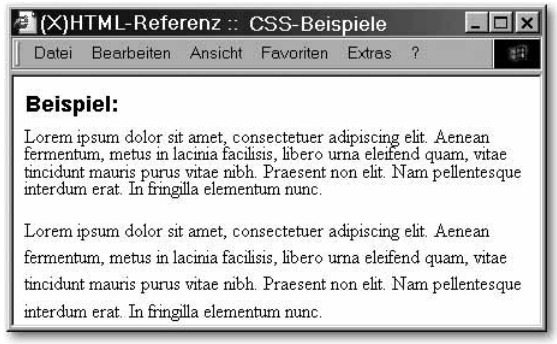

Abbildung 19.13 Zeilenhöhenangaben mit Faktor 0.9 (oben) und 140 % (unten)

text-align

CSS-Level	CSS 1
Anwendung	Blockelemente
Vererbung	ja
Werte	*left, right, center, justify, inherit*
Defaultwert	*inherit*

Das `text-align`-Property bestimmt die horizontale Ausrichtung des Textes innerhalb des Elementcontainers, wobei neben Links- und Rechtsausrichtung auch Blocksatz (*justify*) und Zentrierung möglich sind.

[zB] *Beispiel:*

```
div.center { text-align: center }
```

text-decoration

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*none, blink, underline, overline, line-through, inherit*
Defaultwert	*inherit*

Das `text-decoration`-Property legt die Präsentation des Textinhalts des Containers als durchgestrichen, unter- oder überstrichen sowie blinkend fest. Der Wert `none` entfernt jegliche Defaultpräsentation, wird also eingesetzt, um die Unterstreichung von Hyperlinktext zu unterbinden.

Es können auch durch Leerzeichen getrennt mehrere Präsentationsvorschriften gleichzeitig angeordnet werden. Bei der Anwendung auf Hyperlink-Container genügt jedoch die Zuweisung einer beliebigen Präsentation (außer `underline`), um den Defaultunterstrich zu entfernen. Eine `line-through`-Anweisung addiert sich daher beispielsweise nicht zum HTML-Unterstrich, sondern ersetzt diesen.

Anmerkung: Vom Einsatz des Wertes `"blink"` sollte man absehen (er wird im Internet Explorer auch als CSS-Anweisung ohnehin nicht unterstützt).

Beispiel: **[zB]**

```
a[href] { text-decoration: underline }

<p style="text-decoration:overline">Lorem...</p>

<p style="text-decoration:underline">Lorem...</p>

<p style="text-decoration:overline underline">Lorem...</p>

<p style="text-decoration:line-through underline"> ...</p>
```

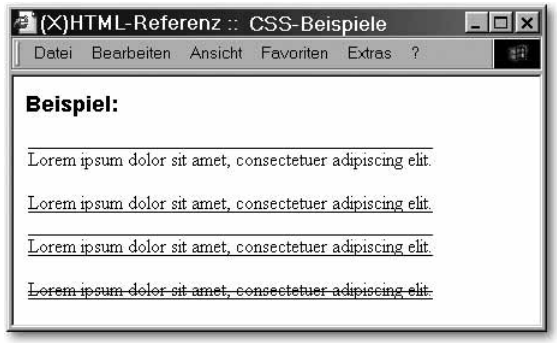

Abbildung 19.14 Text mit Über-, Unter- und Durchstrich sowie Kombinationen

text-indent

CSS-Level	CSS 1
Anwendung	Blockelemente
Vererbung	ja
Werte	*[laenge], [prozent], inherit*
Defaultwert	*0*

Das `text-indent`-Property rückt die erste Zeile des Textinhalts eines Blockelements um den angegebenen Wert ein. Die Einrückung kann durch eine Längenangabe oder einen Prozentwert bezogen auf die Containerbreite angegeben werden.

Negative Werte (»hängender Einzug«) sind möglich, verursachen jedoch ohne weitere Korrektur Darstellungsprobleme (abgeschnittener Text).

[zB] *Beispiel:*

```
p { text-indent: 3em }
```

```
<p style="text-indent:2em">Lorem ipsum ... </p>
```

```
<p style="text-indent:-2em">Lorem ipsum ... </p>
```

Will man einen funktionierenden hängenden Einzug erzeugen, so muss mit `margin-left` gleichzeitig ein entsprechender linker Rand für den Container definiert werden:

```
<p style="text-indent:2em; margin-left:2em">Lorem ... </p>
```

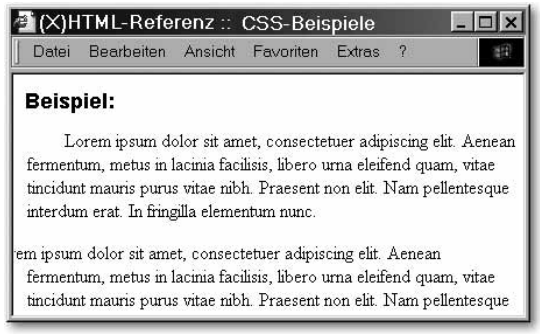

Abbildung 19.15 Positiver und (unkorrigierter) negativer Einzug von 2 em

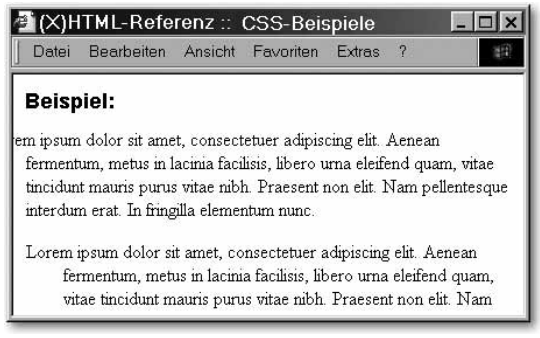

Abbildung 19.16 Unkorrigierter und korrigierter hängender Einzug

text-shadow

CSS-Level	CSS 2; in CSS 2.1 entfallen (geplant für CSS 3)
Anwendung	global
Vererbung	ja
Werte	*[farbe]? [laenge-x] [laenge-y] [laenge-blur]?*
Defaultwert	–

Das `text-shadow`-Property dient dazu, einen Schatteneffekt für den Text des Elements zu definieren. Hierbei kann optional ein Farbwert für den Schatten übergeben werden, ansonsten wird die Textfarbe als Schattenfarbe verwendet. Zwei Längenangaben dienen als Werte für die Verschiebung des Schattens gegenüber dem Text (nach rechts und unten), ein optionaler dritter Längenwert definiert die Unschärfe (Schattengrenze). Für einen Text können gleichzeitig mehrere Schatteneffekte definiert werden, wobei die Wertegruppen durch Komma zu trennen sind.

Anmerkung: Das Property wird derzeit nirgends unterstützt und wurde daher in CSS 2.1 wieder aus der Spezifikation entfernt. Eine Wiedereinführung in CSS 3 ist jedoch zu erwarten.

Beispiel: **[zB]**

```
/* schwarzer Schatten (Default): */
h1 {text-shadow: 0.2em 0.2em}

/* roter Schatten mit 5px Blur: */
h2 {text-shadow: 3px 3px 5px red}

/* roter Schatten ohne Blur überlagert mit gelb, 2px Blur */
h2 {text-shadow: 3px 3px red, yellow -3px 3px 2px}
```

text-transform

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*capitalize, lowercase, uppercase, none, inherit*
Defaultwert	*inherit*

Das `text-transform`-Property bestimmt die Präsentation des Textinhalts des Containers in einer festzulegenden Schreibweise.

- `text-transform: capitalize`
  Den Anfangsbuchstaben jedes Wortes als Großbuchstabe präsentieren

- `text-transform: lowercase`
  Den gesamten Text mit Kleinbuchstaben präsentieren
- `text-transform: uppercase`
  Den gesamten Text mit Großbuchstaben präsentieren
- `text-transform: none`
  Den Text unverändert präsentieren

Es ist lediglich die Präsentation des Textes betroffen. Der Text selbst verbleibt also in der ursprünglichen Schreibweise (dies macht sich beispielsweise bei einer Kopie in die Zwischenablage bemerkbar).

Anmerkung: Die Darstellung von Text in *Kapitälchen* wird mit dem Property `font-variant` und dessen Wert `small-caps` erreicht.

[zB] *Beispiel:*

```
p { text-transform: lowercase }

p { text-transform: uppercase }
```

Abbildung 19.17 Präsentation mit text-transform:lowercase und uppercase

unicode-bidi

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*normal, embed, bidi-override, inherit*
Defaultwert	*inherit*

Das `unicode-bidi`-Property bestimmt, wie bidirektionale Textinhalte zu behandeln sind. Es ist in Zusammenhang mit dem `direction`-Property zu betrachten.

- `unicode-bidi: normal`
  Normaler Textfluss innerhalb des Elements
- `unicode-bidi: embed`
  Inlineelemente innerhalb des Containers werden gemäß des bidirektionalen Algorithmus eingebettet, wobei das `direction`-Property die Textrichtung bestimmt.
- `unicode-bidi: bidi-override`
  Der bidirektionale Algorithmus wird für den Elementinhalt außer Kraft gesetzt.

Anmerkung: Eine ausführliche Erläuterung des bidirektionalen Algorithmus finden Sie in der Referenz der HTML-Elemente bei den Erklärungen zum Element `<bdo>`.

vertical-align

CSS-Level	CSS 1
Anwendung	Inlineelemente, `<td>` und `<th>` sowie `display:table-cell`
Vererbung	nein
Werte	*baseline, bottom, middle, sub, super, text-bottom, text-top, top, [prozent], [laenge*], inherit* * nur CSS2
Defaultwert	*inherit*

Das `vertical-align`-Property bestimmt die vertikale Ausrichtung des Inlinecontainers innerhalb des durch das umliegende Blockelement zur Verfügung stehenden Raumes. Die Ausrichtung wird mit Schlüsselwörtern bestimmt oder einer Prozentangabe, die sich auf die Zeilenhöhe bezieht. Seit CSS 2 ist auch eine Längenangabe möglich, die sich in diesem Fall auf die Textgrundlinie bezieht, wobei positive oder negative Angabe das Element über oder unter die Grundlinie der Textzeile verschiebt.

- `vertical-align: sub`
  Der Container wird zur umgebenden Zeile tiefgestellt (der Text jedoch nicht verkleinert) und dabei mit der Oberkante unter die Zeilenmitte gesetzt.
- `vertical-align: super`
  Der Container wird zur umgebenden Zeile hochgestellt (der Text jedoch nicht verkleinert) und dabei auf die Zeilenmitte gesetzt.
- `vertical-align: baseline`
  Der Container wird an der Grundlinie der umgebenden Zeile ausgerichtet.

- `vertical-align: top`
  Der Container wird am oberen Rand des Elternelements ausgerichtet.
- `vertical-align: bottom`
  Der Container wird am unteren Rand des Elternelements ausgerichtet.
- `vertical-align: middle`
  Der Container wird vertikal im Elternelement zentriert bzw. an der vertikalen Zeilenmitte orientiert (gleicht im Zeilenkontext daher `vertical-align:sub`).
- `vertical-align: text-top`
  Der Container wird am oberen Textrand der umgebenden Zeile ausgerichtet.
- `vertical-align: text-bottom`
  Der Container wird am unteren Textrand der umgebenden Zeile ausgerichtet.

Anmerkung: Die Werte `sub`, `super`, `text-bottom` und `text-top` sind für Tabellenzellen oder mittels `display:table-cell` als Tabellenzelle präsentierte Elemente nicht definiert.

[zB] *Beispiel:*

```
td { vertical-align: bottom }
```

```
<p>Dies ist
 <span style="vertical-align:sub">tiefgestellter</span>
 Text.</p>
```

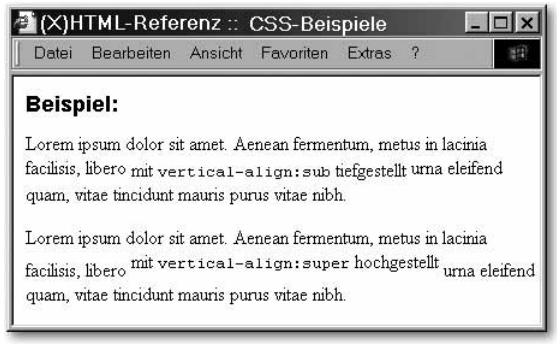

Abbildung 19.18 Mit vertical-align tief- und hochgestellter Text

white-space

CSS-Level	CSS 1
Anwendung	Blockelemente
Vererbung	ja

Werte	*normal, pre, nowrap, inherit*
Defaultwert	*inherit*

Das `white-space`-Property legt die Behandlung von Weißraumzeichen (Zeilenumbruch, Tabulator, Leerzeichen) im betreffenden Elementcontainer fest.

- `white-space: normal`
  Normale Handhabung von Leerraum durch den Browser. Es erfolgt ein automatischer Zeilenumbruch sobald der Text den Rand des Elementcontainers erreicht.
- `white-space: pre`
  Die Normalisierung des Leerraumes durch den Browser wird deaktiviert, alle Leerzeichen, Tabulatoren und Zeilenumbrüche also dargestellt, wie sie im Quelltext vorliegen. Es erfolgt kein automatischer Zeilenumbruch. Das Verhalten entspricht somit dem des HTML-`<pre>`-Containers.
- `white-space: nowrap`
  Die Normalisierung des Leerraumes durch den Browser erfolgt wie gewohnt; es wird lediglich der automatische Zeilenumbruch unterbunden.

Anmerkung: Die beiden weiteren in CSS 2.0 vorgesehenen Werte `pre-wrap` und `pre-line` sind in CSS 2.1 aus der Spezifikation wieder entfernt worden.

Beispiel: **[zB]**

```
pre { white-space: pre }

p   { white-space: normal }

td[nowrap] { white-space: nowrap }
```

word-spacing

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*normal, [laenge], inherit*
Defaultwert	*normal*

Das `word-spacing`-Property legt den Wortabstand zwischen einzelnen Worten eines Elementinhalts fest. Der Wert `normal` präsentiert den Elementinhalt mit unverändertem Wortabstand. Eine Längenangabe erzwingt einen bestimmten Abstand zwischen einzelnen Worten, wobei jedoch der durch das trennende Leerzeichen festgelegte Mindestabstand nicht unterschritten werden kann.

[zB] *Beispiel:*

```
h1 { word-spacing: 0.5em }

p  { word-spacing:0.2em; }
```

19.5.2 Fontproperties

font

CSS-Level	CSS 1
Anwendung	global
Typ	Sammelproperty
Vererbung	ja
Werte	*[font-style] [font-variant] [font-weight] [font-size(/line-height)?] [font-family]*
Defaultwert	*inherit*

Das Sammelproperty `font` bietet die Möglichkeit, alle sechs zur Schriftpräsentation dienenden Properties gemeinsam festzulegen. Es sind, getrennt durch Leerzeichen, die Werte für die Subproperties `font-style`, `font-variant`, `font-weight`, `font-size`/`line-height` und `font-family` anzugeben. Soll eine der Subproperties nicht näher bestimmt werden, so kann an Stelle eines Wertes das Schlüsselwort `inherit` übergeben werden.

Ein Wert für `line-height` ist stets zusammen mit `font-size` anzugeben, wobei zwischen beiden ein trennender Slash stehen muss: `font-size/line-height`.

[zB] *Beispiel:*

```
p {font: 12pt/14pt sans-serif}

p {font: 80% sans-serif}

p {font: x-large/110% "Times New Roman", serif}
```

font-family

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*schriftname [schriftname, ...], inherit*
Defaultwert	*Browserdefault*

Das Property `font-family` bestimmt die Schriftart (Font), in der der Elementinhalt präsentiert werden soll. Es dürfen mehrere Fontnamen durch Komma getrennt übergeben werden, wobei der Browser zunächst versucht, den als ersten übergebenen Font anzuwenden. Ist dieser nicht verfügbar, weicht er auf die folgende Angabe aus usw.

Als letzte Schriftfamilie sollte daher möglichst ein allgemeiner Bezeichner wie `sans-serif` (Serifenlose Schrift), `serif` (Serifenschrift), `fantasy` (Schmuck- oder Symbolschriften), `cursive` (Kursivschriften) oder `monospace` (Schrift konstanter Breite) übergeben werden.

Beispiel: **[zB]**

```
body {font-family: Arial, Helvetica, sans-serif}
```

font-size

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*xx-small, x-small, small, medium, large, x-large, xx-large. [laenge], [prozent], larger, smaller, inherit*
Defaultwert	*medium*

Das `font-size`-Property legt die Größe fest, in der die Schrift des Elements präsentiert werden soll. Hierbei können Größenangaben mit Schlüsselwörtern oder als Längenangabe in verschiedenen Maßeinheiten oder Prozentangaben relativ zur Schriftgröße des umliegenen Containers gemacht werden.

Die Angaben `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large` bestimmen eine absolute Schriftgröße von sehr klein bis sehr groß, die jedoch vom Browser nachskaliert werden kann. Die Darstellungsgröße ist browserabhängig, wobei `small` in der Regel der HTML-Angabe `size="2"`, die Größe `xx-large` der HTML-Angabe `size="6"`entspricht.

Die Werte `smaller` und `bigger` verkleinern bzw. vergrößern die Schrift gegenüber der Schriftgröße des umliegenden Containers um einen Faktor. Die Wirkung entspricht der HTML-Angabe `size="-1"` bzw. `size="+1"`.

Längenangaben können in absoluten Einheiten wie `pt` (Punkt), `pc` (Pica), `mm`, `cm` oder `in` (Zoll) erfolgen, deren tatsächliche Darstellungsgröße jedoch abhängig von der Bildschirmseinstellung ist. Vorzuziehen sind relative Längenangaben in `em` und `en`. Eine Angabe in `px` (Pixel) ist ebenfalls abhängig von der Bildschirmauflösung des Nutzers und gefährdet zudem in manchen Browsern (Internet

Explorer) die Nachskalierbarkeit der Schriftgröße – dies ist deshalb zu vermeiden.

[zB] *Beispiel:*

```
blockquote { font-size: larger }
body { font-size:smaller; }
p    { font-size: 12pt; }
em   { font-size: 150% }
i    { font-size: 1.5em }
```

font-size-adjust

CSS-Level	CSS 2 (in CSS 2.1 entfernt)
Anwendung	global
Vererbung	ja
Werte	
Defaultwert	*inherit*

Das `font-size-adjust`-Property ist dazu gedacht, die x-Höhe der Schrift innerhalb des Elementcontainers an die x-Höhe der Schrift des umliegenden Containers anzupassen, sofern unterschiedliche Schriften verwendet werden. Hintergedanke sind Unterschiede in der Darstellungsgröße zweier Schriften bei gleicher eingestellter Schriftgröße, wodurch sich bei stark abweichender x-Höhe Probleme mit der Lesbarkeit ergeben können.

Als Ausgangspunkt und Maß für die Lesbarkeit dient der so genannte Aspektwert einer Schrift, das Verhältnis von Schrifthöhe zur Höhe der Kleinbuchstaben (x-Höhe). Besitzt eine Schrift einen kleineren Aspektwert als die Bezugsschrift, so sollte sie mittels `font-size-adjust` vergrößert werden, ansonsten verkleinert.

Der zu übergebende Wert entspricht dem **Verhältnis der Aspektwerte** beider Schriften. Für Verdana mit 0.58 und Times New Roman mit 0.46 berechnet sich der Anpassungsfaktor zu 0.58/0.46, also zu 1.26.

[zB] *Beispiel:*

Von Verdana (0.58) nach Comic Sans MS (0.54):

```
font-size-adjust: 1.07
```

Von Verdana (0.58) nach Georgia (0.50):

```
font-size-adjust: 1.16
```

Von Verdana (0.58) nach Times New Roman (0.46):

```
font-size-adjust: 1.26
```

Anmerkung: Das `font-size-adjust`-Property wird derzeit von keinem Browser unterstützt und wurde daher aus der Spezifikation CSS 2.1 wieder entfernt.

font-stretch

CSS-Level	CSS 2 (in CSS 2.1 entfernt)
Anwendung	global
Vererbung	ja
Werte	*ultra-condensed, extra-condensed, condensed, semi-condensed, normal, semi-expanded, expanded, extra-expanded, ultra-expanded, inherit*
Defaultwert	*inherit*

Das `font-stretch`-Property streckt oder staucht die Zeichenabstände innerhalb eines Textes.

Beispiel: **[zB]**

```
body { font-stretch:normal; }

p    { font-stretch:extra-condesed; }

p    { font-stretch:semi-expand; }
```

Anmerkung: Das `font-stretch`-Property wird derzeit von keinem Browser unterstützt und wurde daher aus der Spezifikation CSS 2.1 wieder entfernt.

font-style

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*normal, italic, oblique, inherit*
Defaultwert	*normal*

Das `font-style`-Property bestimmt den Darstellungsstil der Schrift als `italic` (kursiv) oder `oblique` (künstlich kursiv, »schief«, *slanted*). Ebenso kann ein ererbter kursiver Darstellungsstil auf `normal` zurückgesetzt werden.

- `font-style: normal`
  ordnet normale, nicht-kursive Darstellung an

- `font-style: italic`
  ordnet die Verwendung eines kursiven Schriftschnittes an. Steht dieser nicht zur Verfügung, kann der Browser auf einen künstlich schräggestellten Schnitt (`oblique`) ausweichen.
- `font-style: oblique`
  ordnet die Verwendung eines künstlich schräggestellten Schriftschnittes (`oblique`) an. In der Regel werden Browser dennoch echten kursiven Schnitt einsetzen, sofern ein solcher für die vorliegende Schrift zur Verfügung steht.

[zB] *Beispiel:*

```
h1, h2, h3 { font-style: italic }
```

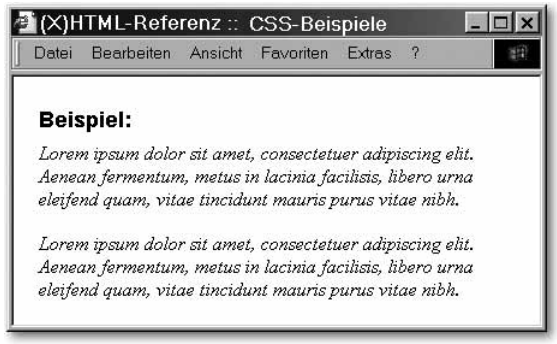

Abbildung 19.19 Kein Unterschied zwischen italic (oben) und oblique (unten)

font-variant

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*normal, small-caps, inherit*
Defaultwert	*normal*

Das `font-variant`-Property dient mit dem Wert `small-caps` dazu, einen Text in Kapitälchen darzustellen, d.h. die Kleinbuchstaben durch verkleinerte Großbuchstaben zu ersetzen. Der Wert `normal` setzt die Darstellung auf normale Präsentation zurück, sofern die Kapitälchendarstellung von einem umgebenden Container ererbt wurde.

[zB] *Beispiel:*

```
p { font-variant: small-caps }
```

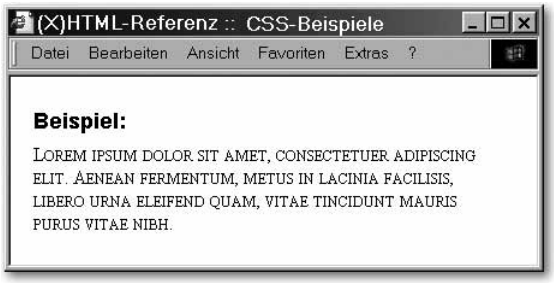

Abbildung 19.20 Darstellung eines Textabschnittes mit Kapitälchen

font-weight

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*normal*, *bold*, *bolder*, *lighter*, *100*, *200*, *300*, *400*, *500*, *600*, *700*, *800*, *900*, *inherit*
Defaultwert	*normal*

Das `font-weight`-Property bestimmt in einer am Desktop-Publishing orientierten Weise den **Fettegrad** (*weight*) der Schrift des erfassten Containers. In der Theorie kann zwar ein abgestufter Fettegrad angegeben werden, in der Praxis liegen (bei Bildschirmausgabe) dem Browser für die meisten Schriften jedoch nur zwei alternative Schriftschnitte (normal und fett) vor.

Dies führt dazu, dass für die Skalierungsangaben 100 bis 900 (von mager bis extrafett) meist nur der Wert 500 als Schwellenwert verwendet wird (für alle Werte unter 500 wird der Schnitt »normal« eingesetzt, ab 500 der Schnitt »fett«). Analog bewirken die Optionen *bolder* (fetter) und *lighter* (magerer) keine Herauf- oder Herabstufung um einen Fettegrad, sondern gegebenenfalls nur ein Umschalten zur jeweils anderen Variante. Wird bereits der Fettegrad »fett« verwendet, zeigt *bolder* jedoch keine Wirkung, ebenso wie *lighter* bei bereits eingesetztem normalem Schriftschnitt keinen Effekt hat.

Beispiel: **[zB]**

```
p { font-weight: normal }      /* entspricht "400"        */
h1 { font-weight: 700 }        /* entspricht "bold"       */
body { font-weight: 400 }      /* keine sichtbare Wirkung */
strong { font-weight: bolder } /* wird nicht "fetter"     */
```

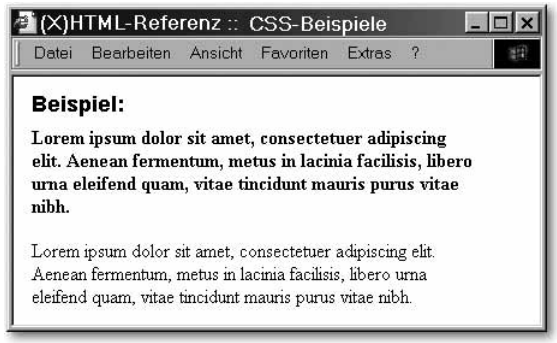

Abbildung 19.21 Fettheitsangabe mit font-weight: 700 (oben) und 400 (unten)

19.5.3 Farben und Hintergründe

CSS kennt innerhalb eines Containers eine Vordergrund- bzw. Textfarbe (`color`) und eine Hintergrundfarbe (`background-color`). Letzte stellt eine Untereigenschaft der Hintergrundformatierung dar, die insgesamt mittels des `background`-Properties vorgenommen werden kann. Alternativ kann jede einzelne Untereigenschaft auch einzeln angesprochen werden.

background

CSS-Level	CSS 1
Anwendung	global
Typ	Sammelproperty
Vererbung	nein
Werte	*[background-color] [background-image] [background-attachment] [background-repeat] [background-position]*
Defaultwert	–

Das Property `background` dient als **Sammelproperty** für die Bestimmung der Eigenschaften des Hintergrunds eines Containers.

Mittels `background` können sowohl die Hintergrundfarbe (`background-color`) als auch das Hintergrundbild (`background-image`) und dessen Eigenschaften festgelegt werden. In diesem Zusammenhang kann angegeben werden, ob das Hintergrundbild gegenüber dem Browserfenster fixiert werden soll (`background-attachment`), ob und in welchen Richtungen es gekachelt werden soll (`background-repeat`) und wo es positioniert wird (`background-position`).

Anmerkung: Werden gleichzeitig Hintergrundfarbe und -bild angegeben, so verdeckt das Hintergrundbild die Hintergrundfarbe. Die Hintergrundfarbe wird

jedoch bei teiltransparenten Hintergrundgrafiken oder nicht deckender Kachelung (`no-repeat`, `repeat-x`, `repeat-y`; siehe background-repeat) sichtbar.

Beispiel: **[zB]**

```
/* Hintergrundfarbe rot: */
body { background: red }

/* Hintergrundfarbe teilweise sichtbar: */
body { background: #ff9999 url('bg.gif') repeat-y; }

/* positioniertes Hintergrundbild; s.  Abb. */
body { background:url('bg.gif') 100px 25px no-repeat; }
```

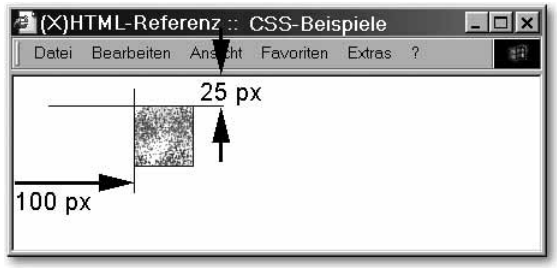

Abbildung 19.22 Nicht wiederholtes, positioniertes Hintergrundbild

background-attachment

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*scroll, fixed, inherit*
Defaultwert	*scroll*

Das `background-attachment`-Property legt fest, ob das Hintergrundbild gegenüber dem Browserfenster fixiert wird, also beim Scrollen des Dokuments stationär bleibt.

- `scroll` (Default)
  Das Hintergrundbild scrollt in Zusammenhang mit dem Seiteninhalt.
- `fixed`
  Das Hintergrundbild ist gegenüber dem Browserfenster fixiert.

Beispiel: **[zB]**

```
body {
      background: #ff9999 url("bg.gif");
```

```
background-repeat: repeat-y;
background-attachment: fixed;
}
```

background-color

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[farbname], [farbwert], transparent, inherit*
Defaultwert	*transparent*

Das Property `background-color` bestimmt die Hintergrundfarbe des Containerinhaltsbereichs eines Elements. Handelt es sich um den `<body>`-Container, ist dies die Hintergrundfarbe des gesamten Dokuments, die an die in diesem enthaltenen Container weitervererbt wird.

Handelt es sich um einen Blockcontainer, so gilt die Hintergrundfarbe gemäß des Boxmodells für den Textbereich, den Bereich des diesen umgebenden Innenabstands (*padding*) und den Rahmenbereich (*border*), sofern Letzterer auf `"transparent"` gesetzt ist.

Für Inlinecontainer gilt die Hintergrundfarbe für den Bereich des Textinhalts zzgl. Padding (das Boxmodell greift in diesem Fall nicht vollständig).

[zB] *Beispiel:*

```
/* abgekürzte Farbwertsyntax: */
h1 { background-color: #F00 }

p { background-color: transparent; }

body { background-color: #dddddd; }

p.hellgrau {
           background-color:#eeeeee;
           padding:6px;
           }

span.grau {
           background-color:#999999;
           padding:6px;
           border:solid 2px white;
        }
```

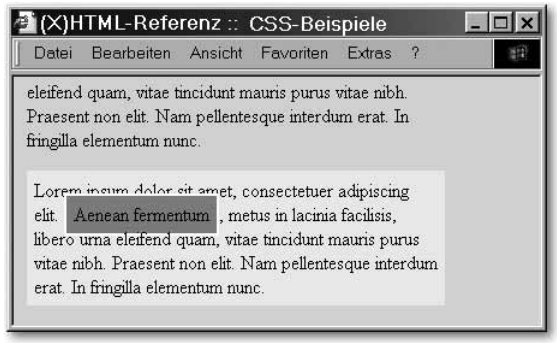

Abbildung 19.23 Hintergrundfarbe bei Body, Block- und Inlineelement

background-image

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*url([*pfad/dateiname*]), none, inherit*
Defaultwert	*none*

Das `background-image`-Property weist den Hintergrundbereich eines Elementcontainers (Textbereich, Innenabstand und Rahmenbereich) ein Hintergrundbild zu. Wird nicht gleichzeitig mit `background-repeat` und `background-position` gearbeitet, so kachelt sich das Hintergrundbild ab der linken oberen Ecke des Inhaltsbereiches des Containers.

Anzugeben ist der URL der einzubindenden Grafik oder der Wert `"none"`, der die Darstellung eines Hintergrundbildes explizit untersagt.

Vorsicht bei Pfadangaben in externen CSS-Dateien

Bei der URL-Angabe ist zu beachten, dass dieser sich auf den Basis-URI der CSS-Regel bezieht, sofern eine relative Pfadangabe erfolgt. Für einen Inline- oder Dokumentstyle entspricht dies dem Basis-URI des HTML-Dokuments, *nicht jedoch* für eine externe CSS-Datei. Dies gilt ebenfalls für importierte Stylesheetregeln, auch wenn diese ansonsten nominell den Dokumentstyles zugeschlagen werden.

Beispiel: **[zB]**

```
body { background-image: url("bg.gif") }

p { background-image: none }

p { background-image: url('schrauben.jpg'); padding:15px; }
```

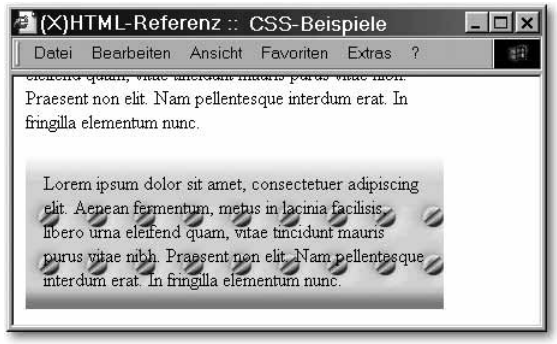

Abbildung 19.24 Ein Textabsatz mit Hintergrundbild

background-position

CSS-Level	CSS 1
Anwendung	Blockelemente
Vererbung	nein
Werte	*[prozent], [laenge], [prozent laenge], [laenge prozent], [laenge laenge],[prozent prozent], top left, top right, bottom left, bottom right, center, inherit*
Defaultwert	*0 % 0 %*

Das `background-position`-Property bezieht sich auf die Position eines mit `background` oder `background-image` eingebundenen Hintergrundbildes (den Wirkungsbereich einer Hintergrundfarbe zu beeinflussen, ist nicht möglich).

Das Property übernimmt Längen- oder Prozentangaben oder Schlüsselwörter, die einzeln oder paarweise kombiniert auftreten können.

- **Längenangaben**
  Längenangaben treten in der Regel paarweise auf, wobei sich der erste Wert auf die horizontale Verschiebung der Ursprungskoordinate des Bildes bezieht, der zweite auf deren vertikale Verschiebung. Entfällt der zweite Wert, so wird die Grafik vertikal zentriert. Bei horizontaler oder vertikaler Kachelung der Hintergrundgrafik füllt das Hintergrundbild den Bildschirm in jedem Falle vollständig in der angegebenen Richtung; eine Verschiebung in gleicher Richtung bewirkt lediglich einen Offset der Reihe. Negative Werte verschieben das Bild aus dem Containerbereich nach außen.
- **Prozentangaben**
  Auch Prozentangaben treten normalerweise paarig auf. Der erste Wert bezieht sich auch hier auf die horizontale, der zweite auf die vertikale Position

der Grafik. Soll vertikal zentriert werden, so kann der zweite Wert entfallen. Die Angaben beziehen sich zum einen auf die Abmessungen des Elternelements, also den zur Verfügung stehenden Raum. Anders als Längenangaben orientieren sie sich zum anderen jedoch nicht am Koordinatenursprung der Grafik, sondern an einem Punkt im Inneren der Grafikfläche, der sich wiederum (spiegelbildlich) aus den Prozentangaben ergibt. Auch hier verschieben negative Werte das Bild aus dem Containerbereich nach außen.

Beispiel: Gibt man eine Verschiebung von 100 % 100 % an, so ist als Zielpunkt die rechte untere Ecke des Containers, der Deckungspunkt der Grafik wird bei 100 % ihrer Breite und 100 % ihrer Höhe lokalisiert, entspricht also ebenfalls deren rechter unterer Ecke. **[zB]**

Bei einer Verschiebung von `50 % 50 %` liegt der Zielpunkt in der Mitte des Containers, der Deckungspunkt der Grafik analog in deren geometrischer Mitte (siehe Abbildung 19.25).

- **Schlüsselwörter**
  Die Schlüsselwörter gleichen in ihrem Verhalten festen Prozentangaben, wobei `left` und `top` dem Wert 0 % in der Horizontalen bzw. Vertikalen entsprechen, `right` und `bottom` hingegen dem Wert 100 % in der Horizontalen und Vertikalen. Der Wert `center` entspricht 50 % und kann für jede Verschiebungsrichtung sowie auch alleinstehend für beide gleichzeitig verwendet werden.

  Beispiel: Die Angabe `top right` entspricht der Verschiebung `0 % 100 %` und rückt das Hintergrundbild an die rechte obere Ecke des Containers. **[zB]**

  Die Angabe `bottom left` wiederum entspricht dem Wertepaar `100 % 0 %` und verschiebt die Hintergrundgrafik an die untere, linke Ecke.

  Die Einzelangabe `right` ohne Angabe einer vertikalen Verschiebung entspricht dem Wertepaar `100 % 50 %`, da automatisch vertikal zentriert wird.

  Beispiel: **[zB]**

```
body { background: url("bg.gif") 50 150 }
body { background: url("bg.gif") center }
body { background-position right top }
body { background-position: 27% 3cm; }
```

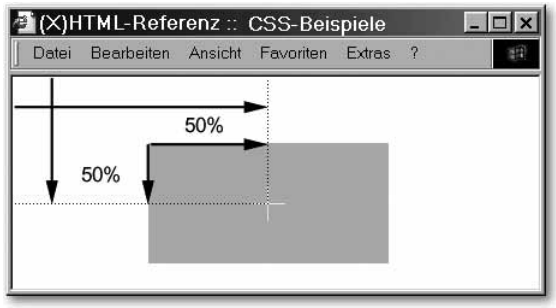

Abbildung 19.25 Zentrierung eines Hintergrundbildes mit Prozentangaben

[zB] *Beispiel:* Ein einzelnes Hintergrundbild, in einem Textabsatz links unten positioniert, bewirkt die Darstellung mit »runder Ecke«. Hierfür wird ein Teil der Grafik in der Farbe des Seitenhintergrundes gefärbt (hier weiß), der andere (die runde Ecke) erhält die Hintergrundfarbe des Textabsatzes:

```
<p style="background:#999999 url('ecke_unten_links.gif')
 no-repeat; background-position:bottom left; padding:10px;">
 Lorem ipsum dolor sit amet... </p>
```

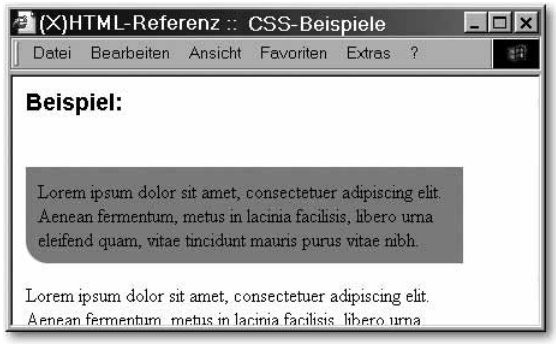

Abbildung 19.26 Textabsatz mit Hintergrundbild »bottom left«

background-repeat

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*repeat, no-repeat, repeat-x, repeat-y, inherit*
Defaultwert	*repeat*

Das Property `background-repeat` bestimmt, ob und in welche Richtung(en) ein Hintergrundbild innerhalb der Containerfläche wiederholt (gekachelt) wird. Das Property kann gleichzeitig mit den anderen `background`-Properties eingesetzt werden.

- `background-repeat:` **repeat** (Default)
  Das Hintergrundbild wird horizontal und vertikal wiederholt und innerhalb des gesamten Hintergrundbereichs gekachelt.
- `background-repeat:` **no-repeat**
  Das Hintergrundbild wird nur einmal an der durch `background-position` angegebenen Position angezeigt. Ist die Inhaltsfläche des Elements größer als das vorgesehene Hintergrundbild, wird der restliche Bereich mit der Hintergrundfarbe aufgefüllt.
- `background-repeat:` **repeat-x**
  Das Hintergrundbild wird horizontal wiederholt angezeigt. Ist das `background-position`-Property gesetzt, wird nur dessen zweiter Wert verwendet, um die Hintergrundbilder vertikal zu verschieben.
- `background-repeat:` **repeat-y**
  Das Hintergrundbild wird vertikal wiederholt angezeigt. Ist das `background-position`-Property gesetzt, wird nur dessen erster Wert verwendet, um die Hintergrundbilder horizontal zu verschieben.

Beispiel: **[zB]**

```
body { background-repeat: repeat-y; }
```

Abbildung 19.27 Vertikale Wiederholung des Hintergrundbildes mit repeat-y

Beispiel: **[zB]**

```
body { background-repeat: repeat-x; }
```

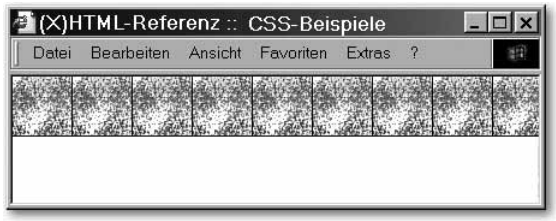

Abbildung 19.28 Horizontale Wiederholung des Hintergrundbildes mit repeat-y

color

CSS-Level	CSS 1
Anwendung	global
Vererbung	ja
Werte	*[farbname], [farbwert], inherit*
Defaultwert	–

Das `color`-Property bestimmt die Vordergrundfarbe des Containerinhalts, mit anderen Worten die Textfarbe. Als Wert kann ein Farbname oder ein Farbwert in der für CSS üblichen Notierung eingesetzt werden.

[zB] *Beispiel:*

```
h1  { color: '#ff0000'; }    /* "rot" (Hexwert)  */
span.rot  { color: red }     /* "rot" (Farbname) */
em  { color: rgb(255,0,0) }  /* "rot" (RGB-Wert) */
body { background: red; color: white; }
```

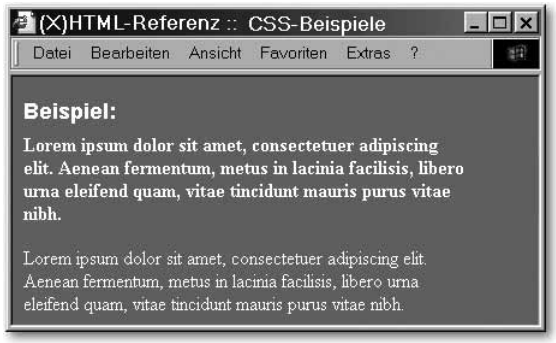

Abbildung 19.29 Weiße Schrift auf rotem Hintergrund

19.5.4 Boxproperties

Das CSS-Boxmodell ist im Besonderen auf **Blockelemente** anwendbar – für Inlineelemente ist es nur eingeschränkt gültig. Aus der Sicht von CSS wird jeder Blockcontainer als »Box« betrachtet, die in vier Bereiche (Inhalt, Padding, Border, Margin) eingeteilt werden kann.

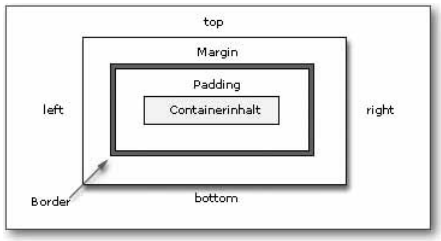

Abbildung 19.30 Die CSS-Box mit Inhalt, Padding, Border und Margin

- **Containerinhalt**
  Die Größe dieses Bereichs ist durch die Abmessungen des Inhalts festgelegt, beispielsweise also Textmenge und Zeilenhöhe, wie sie sich aus schrifttypischen Ober- und Unterlängen ergibt.
- **Padding** – Leerraum um den Inhalt
  Der `padding`-Bereich (Innenabstand) entspricht dem Cellpadding in HTML-Tabellenzellen, gehört also zum inneren Bereich bzw. Hintergrund des Elements. Die Maße des Paddings sind für alle vier Seiten (`top`, `right`, `bottom`, `left`) getrennt wählbar.
- **Border** – ein Rahmen um den Innenraum
  Das `border`-Property definiert einen Rahmen um den Innenraum (Containerinhalt mit Padding), der selbst ebenfalls noch zum Inhaltsbereich zählt. Breite, Farbe und Erscheinungsbild des Rahmens sind für alle vier Seiten getrennt wählbar.
- **Margin** – der Leerraum um das Element
  Der Marginbereich (Außenabstand) bestimmt den Abstand zu Nachbarcontainern. Der Außenabstand kann für jede Seite einzeln festgelegt werden. Er zählt nicht zum Inhaltsbereich des Containers, ist also nicht über das `background-color`-Property einzufärben (vielmehr »erbt« er die Hintergrundfarbe des umgebenden Blocks).

Inhaltsbereich des Containers

Die Bereiche »Containerinhalt«, »Padding« und »Border« bilden zusammen den Inhaltsbereich des Containers, auf den Hintergrundfarben und -grafiken angewendet werden.

padding

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[laenge], [prozent], inherit*
Defaultwert	–

Das `padding`-Property legt den Innenabstand des Textbereichs eines Containers von seiner (gedachten) Grenze fest, das so genannte »Padding«. Es können zwischen einem und vier Werten übergeben werden, wobei sowohl Längen- als auch Prozentangaben wie auch eine Mischung aus beidem möglich sind. Prozentangaben beziehen sich auf die Breite des Elternelements. Negative Werte können nicht übergeben werden bzw. werden als Null interpretiert.

Da ein Padding normalerweise nicht vererbt wird, muss ein vom Elternelement zu erbender Paddingwert mit `"inherit"` angeordnet werden.

Der Defaultwert ist elementabhängig.

- `padding: [wert]`
  Wird ein Wert übergeben, so bezieht sich dieser auf alle vier Seiten des Containers und weist allen den gleichen Wert zu.
- `padding: [wert] [wert]`
  Werden zwei Werte übergeben, so bezieht sich der erste auf das obere und das untere Padding, der zweite auf das Padding links und rechts.
- `padding: [wert] [wert] [wert]`
  Die Übergabe von drei Werten legt mit dem ersten Wert das obere, mit dem zweiten das linke und rechte, sowie mit dem dritten das untere Padding fest
- `padding: [wert] [wert] [wert] [wert]`
  Vier übergebene Werte legen (in dieser Reihenfolge) das obere, das rechte, das untere und das linke Padding fest.

[zB] *Beispiel:*

```
p { padding: 1em; }
p { padding:0.2em 1.5em }
p { padding:1em 0.5em 0.5em 2em }
```

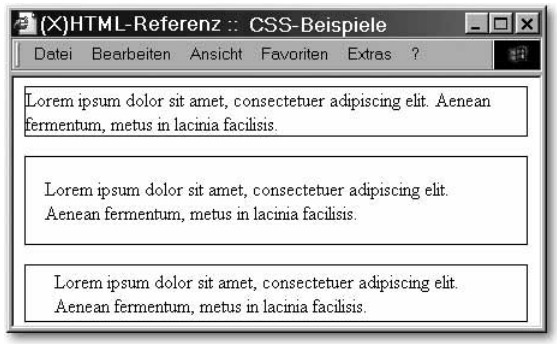

Abbildung 19.31 Textabsatz ohne (oben) und mit verschiedenen Paddingwerten

margin

CSS-Level	CSS 1
Anwendung	Blockelemente; Inlineelemente nur bedingt
Vererbung	nein
Werte	*[laenge], [prozent], inherit*
Defaultwert	*0*

Das `margin`-Property legt den Außenabstand des Inhaltsbereichs eines Containers fest, das so genannte »Margin«. Es können zwischen einem und vier Werten übergeben werden, wobei sowohl Längen- als auch Prozentangaben wie auch eine Mischung aus beidem möglich sind. Prozentangaben beziehen sich auf die Breite des Elternelements. Marginwerte dürfen auch negativ sein und führen auf der entsprechenden Seite gegebenenfalls zu einer Überlagerung mit dem Nachbarcontainer (siehe unterer Textabsatz in Abbildung 19.32).

Da ein Margin ebenso wie ein Padding nicht vererbt wird, muss ein vom Elternelement zu erbender Marginwert mit `"inherit"` angeordnet werden.

Der Defaultwert ist elementabhängig. Textabsätze und Überschriften besitzen beispielsweise einen Defaultmargin nach unten.

- `margin: [wert]`
  Wird ein Wert übergeben, so bezieht sich dieser auf alle vier Seiten des Containers und weist allen den gleichen Wert zu.
- `margin: [wert] [wert]`
  Werden zwei Werte übergeben, so bezieht sich der erste auf das obere und das untere Margin, der zweite auf das Margin links und rechts.

- `margin: [wert] [wert] [wert]`
  Die Übergabe von drei Werten legt mit dem ersten Wert das obere, mit dem zweiten das linke und rechte sowie mit dem dritten das untere Margin fest.
- `margin: [wert] [wert] [wert] [wert]`
  Vier übergebene Werte legen (in dieser Reihenfolge) das obere, das rechte, das untere und das linke Margin fest.

[zB] *Beispiel:*

```
p { margin:0px; }

p { margin:0.2em 1.5em; }

p { margin:-1.2em 0em 0.5em; }
```

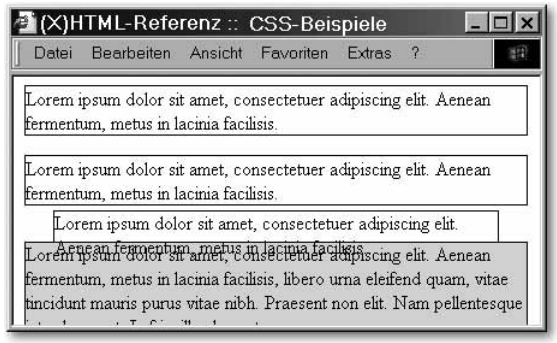

Abbildung 19.32 Textabsätze mit verschiedenen, teilweise negativen Margins

border

CSS-Level	CSS 1
Anwendung	Blockelemente; Inlineelemente nur bedingt
Typ	Sammelproperty
Vererbung	nein
Werte	*[border-style] [border-width] [border-color], inherit*
Defaultwert	*0*

Das Sammelproperty `border` erhält drei Angaben zu Aussehen (`border-style`), Breite (`border-width`) und Farbe (`border-color`) des Rahmens. Die drei übergebenen Werte wirken gleichmäßig auf alle vier Seiten des Rahmens.

[zB] *Beispiel:*

```
<p style="border:solid 1px black">Lorem ipsum ...</p>
```

```
<p style="border:solid 4px #dddddd">Lorem ipsum ...</p>

<p style="border:dotted 1px black">Lorem ipsum ...</p>

<p style="border:dotted 8px #aaaaaa">Lorem ipsum ...</p>
```

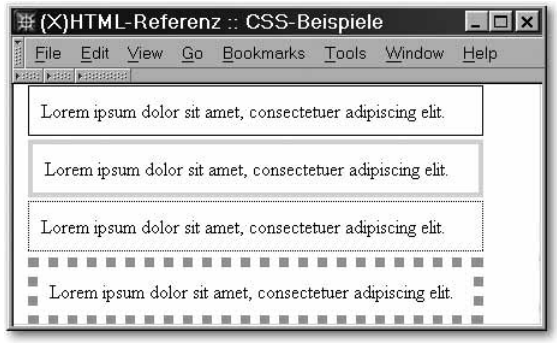

Abbildung 19.33 Rahmen mit verschiedenen Werten (Mozilla)

border-color

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[farbname], [farbwert], transparent, inherit*
Defaultwert	–

Das `border-color`-Property legt die Farbe(n) eines Rahmens fest. Es können in Form von Farbnamen und Farbwerten in Hexadezimal- oder RGB-Schreibweise zwischen einem und vier Werten übergeben werden.

Da Rahmenfarben nicht vererbt werden, müssen vom Elternelement zu erbende Werte mit `"inherit"` angeordnet werden.

Die Defaultrahmenfarbe ist die Vordergrundfarbe des Elements. Wird als Rahmenfarbe insgesamt oder für einen Teilbereich der Wert `"transparent"` zugewiesen, so wirkt an dieser Stelle die Hintergrundfarbe des Containers.

- `border-color: [wert]`
  Wird ein Wert übergeben, so bezieht sich dieser auf alle vier Seiten des Rahmens und weist allen den gleichen Wert zu.
- `border-color: [wert] [wert]`
  Werden zwei Werte übergeben, so bezieht sich der erste auf die obere und untere Rahmenseite, der zweite auf diejenige links und rechts.

- `border-color: [wert] [wert] [wert]`
  Die Übergabe von drei Werten legt mit dem ersten Wert die obere, mit dem zweiten die linke und rechte sowie mit dem dritten die untere Farbe fest.
- `border-color: [wert] [wert] [wert] [wert]`
  Vier übergebene Werte legen (in dieser Reihenfolge) die obere, die rechte, die untere und die linke Rahmenfarbe fest.

[zB] *Beispiel:*

```
<p style="padding:5px; margin:5px; border-style:solid; border-
width:6px; border-color:black;">Lorem ipsum ...</p>

<p style="padding:5px; margin:5px; border-style:solid; border-
width:6px; border-color:black #dddddd;">Lorem ipsum ...</p>

<p style="padding:5px; margin:5px; border-style:solid; border-
width:6px; border-color:black #dddddd #aaaaaa;">Lorem ipsum ...</p>

<p style="padding:5px; margin:5px; border-style:solid; border-
width:6px; border-color:black #999999 #dddddd #aaaaaa;">Lorem ipsum
...</p>
```

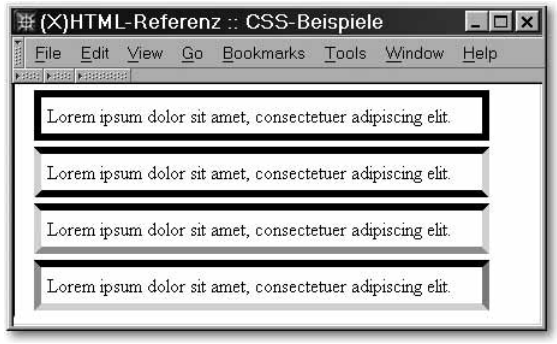

Abbildung 19.34 Rahmen mit einer bis vier Farbangaben

border-style

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*none*, *solid*, *dotted*, *dashed*, *double*, *groove*, *ridge*, *inset*, *outset*, *inherit*
Defaultwert	*none*

Das `border-style`-Property bestimmt das Aussehen (*style*) des Rahmens. Es können aus einer Reihe vordefinierter Schlüsselwörter bis zu vier Werte übergeben werden. Damit das `border-style`-Property Wirkung zeigt, muss mindestens auch ein Wert für die Rahmenbreite `border-size` bestimmt sein (die Rahmenfarbe wird in Abwesenheit einer Vorgabe auf den Vordergrundfarbwert des Containers gesetzt).

Ein einzelner Wert für `border-style` bestimmt den Rahmenstil aller vier Seiten, zwei Werte den Stil oben/unten sowie rechts/links, drei Werte den Stil oben, rechts/links sowie unten, vier Werte den Stil aller vier Seiten von oben an im Uhrzeigersinn gerechnet.

- `border-style: none`
  Es wird keine Rahmenlinie gezeigt
- `border-style: dotted`
  Es wird eine gepunktete Rahmenlinie gezeigt, wobei der Internet Explorer die Linie bei 1px Breite gestrichelt und bei Werten darüber mit runden Punkten darstellt – andere Browser setzen Quadrate für die Punktlinen ein.
- `border-style: dashed`
  Es wird eine gestrichelte Rahmenlinie gezeigt.
- `border-style: solid`
  Es wird eine durchgezogene, je Seite einfarbige Rahmenlinie gezeigt.
- `border-style: double`
  Es wird eine doppelte, je Seite einfarbige Rahmenlinie gezeigt. Die zur Verfügung stehende Rahmenbreite wird gedrittelt und der mittleren Linie die Hintergrundfarbe des Containers zugewiesen (bzw. transparent gesetzt). Der Einsatz von `"double"` ist erst ab einer Rahmenbreite von 3 px sinnvoll.

Beispiel: **[zB]**

```
<p style="border-style:dotted; border-width:3px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:dashed; border-width:3px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:solid; border-width:3px;
 border-color:black;">Lorem ipsum ...</p>

<p style="background-color:#dddddd; border-style:double;
 border-width:3px; border-color:black;">Lorem ipsum ...</p>
```

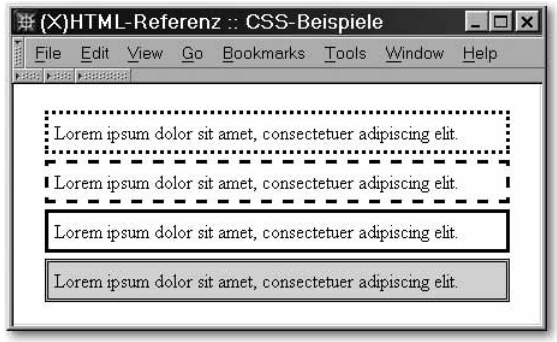

Abbildung 19.35 Rahmen in dotted, dashed, solid und double, je 3 px Breite

Die Werte `"groove"`, `"ridge"` und `"inset"` stellen verschiedene Varianten von 3D-Rahmenlinien dar, die erst ab einer Breite von mindestens zwei, besser vier Pixel sinnvoll anwendbar sind (wenn auch nicht attraktiv[5]). Die je Seite definierte Rahmenfarbe wird nach je einer Licht- und Schattenfarbe hin extrapoliert.

- `border-style: groove`
  Es wird eine mehrfarbige Rahmenlinie mit plastischem 3D-Effekt gezeigt (vertiefte Rahmenlinie).
- `border-style: ridge`
  Es wird eine invertierte mehrfarbige Rahmenlinie mit plastischem 3D-Effekt gezeigt (erhabene Rahmenlinie).

Die Werte `"inset"` und `"outset"` weisen als 3D-Rahmenlinien der oberen und linken sowie der unteren und rechten Rahmenseite jeweils Licht und Schattenfarbe zu, wobei `"inset"` eine Lichtquelle oben links, `"outset"` eine Lichtquelle unten rechts simuliert (d.i. eine erhabene bzw. vertiefte Fläche). Die Werte können ab 1 px Rahmenbreite eingesetzt werden.

- `border-style: inset`
  Es wird ein zweifarbiger Rahmen mit Beleuchtungseffekt oben links gezeigt.
- `border-style: outset`
  Es wird ein zweifarbiger Rahmen mit Beleuchtungseffekt unten rechts gezeigt.

[zB] *Beispiel:*

```
<p style="border-style:groove; border-width:4px;
 border-color:black;">Lorem ipsum ...</p>
```

5 Dies stellt natürlich die sehr subjektive Meinung des Autors dar ;-).

```
<p style="border-style:ridge; border-width:4px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:inset; border-width:1px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:outset; border-width:1px;
 border-color:black;">Lorem ...</p>
```

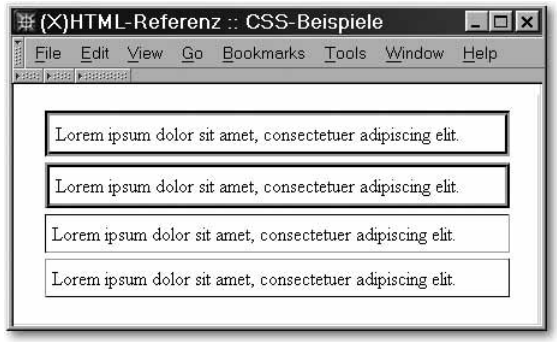

Abbildung 19.36 Borderstyles (von oben) groove, ridge, inset, outset

border-width

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[laenge], thin, medium, thick, inherit*
Defaultwert	*medium*

Das `border-width`-Property bestimmt die Breite der Rahmenlinie. Es können bis zu vier Werte in Form von Längenangaben oder dreier vordefinierter Schlüsselwörter in beliebiger Kombination übergeben werden.

Damit das `border-width`-Property Wirkung zeigt, muss mindestens ein Wert für den Rahmenstil `border-style` bestimmt sein (die Rahmenfarbe wird in Abwesenheit einer Vorgabe auf den `color`-Wert des Containers gesetzt).

Folgende Schlüsselworte können zur Breitenangabe verwendet werden (die Schlüsselworte können auch auf Einzelbreiten angewendet werden):

- `border-width: thin`
  zeichnet eine dünne Rahmenlinie (entspricht 1 px Breite)

- `border-width: medium` (Default)
  zeichnet eine mitteldicke Rahmenlinie (entspricht 3 px Breite)
- `border-width: thick`
  zeichnet eine dicke Rahmenlinie (entspricht 5 px Breite)

[zB] *Beispiel:*

```
<p style=" border-style:solid; border-width:thin;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:solid; border-width:medium;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:solid; border-width:thick;
 border-color:black;">Lorem ipsum ...</p>
```

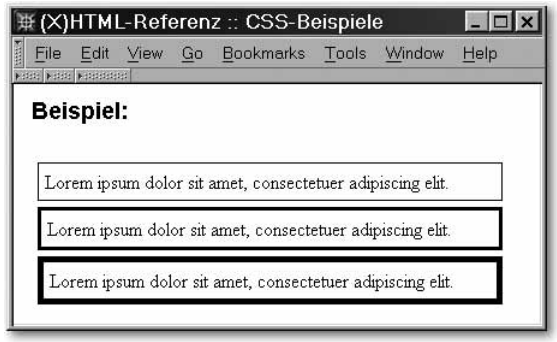

Abbildung 19.37 Rahmenbreite mit (von oben) thin, medium, thick

Mittels Längenwerten können Breiten für jede einzelne Seite vergeben werden:

- `border-width: [laenge]`
  Wird ein Wert übergeben, so bezieht sich dieser auf alle vier Seiten des Rahmens und weist allen die gleiche Breite zu.
- `border-width: [laenge] [laenge]`
  Werden zwei Werte übergeben, so bezieht sich der erste auf die obere und untere Rahmenseite, der zweite auf diejenige links und rechts.
- `border-width: [laenge] [laenge] [laenge]`
  Die Übergabe von drei Werten legt mit dem ersten Wert die obere, mit dem zweiten die linke und rechte sowie mit dem dritten die untere Breite fest
- `border-width: [laenge] [laenge] [laenge] [laenge]`
  Vier übergebene Werte legen, in dieser Reihenfolge, die obere, die rechte, die untere und die linke Rahmenbreite fest.

Beispiel: **[zB]**

```
<p style="border-style:solid; border-width:2px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:solid; border-width:2px 6px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:solid; border-width:2px 6px 12px;
 border-color:black;">Lorem ipsum ...</p>

<p style="border-style:solid; border-color:black;
 border-width:1px 24px 12px 4px; ">Lorem ipsum ...</p>
```

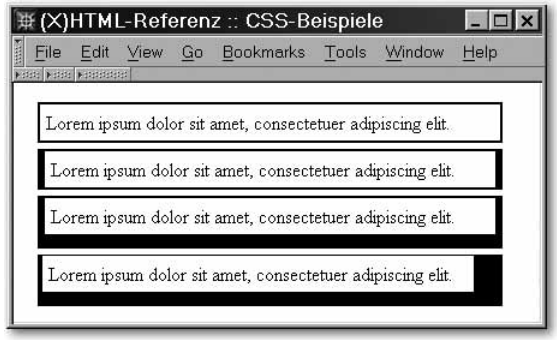

Abbildung 19.38 Rahmenbreite mit ein bis vier Breitenwerten

height

CSS-Level	CSS 1
Anwendung	global (außer Tabellenspalten und -spaltengruppen)
Vererbung	nein
Werte	*[laenge], [prozent], auto, inherit*
Defaultwert	*auto*

Das `height`-Property bestimmt die Höhe eines Elements. Die hier übergebene Abmessung bezieht sich auf den inneren Bereich der Box – eventuell angegebene Margins erhöhen den Platzbedarf des Gesamtgebildes entsprechend.

- `height: auto` (Default)
  setzt die Innenhöhe auf den durch die Höhe des Inhalts erzwungenen Wert
- `height: [laenge]`
  setzt die Innenhöhe auf den übergebenen Wert, auch wenn der Inhalt gerin-

gere oder größere Höhe besitzt. Ist der Inhalt höher, so ragt er nach unten aus dem Containerbereich heraus (sofern nicht `overflow:hidden` angegeben ist).

- `height: [prozent]`
  setzt die Innenhöhe auf den übergebenen Prozentwert, der sich an der Höhe des Elternelements misst, auch wenn der Inhalt geringere oder größere Höhe besitzt. Ist der Inhalt höher, so ragt er nach unten aus dem Containerbereich heraus (sofern nicht `overflow:hidden` angegeben ist).

[zB] *Beispiel:*

```
<p style="height:25px; border-style:solid;
 border-width:1px;border-color:black;">Lorem ipsum... </p>

<p style="height:50px; border-style:solid;
 border-width:1px;border-color:black;">Lorem ipsum .... </p>
```

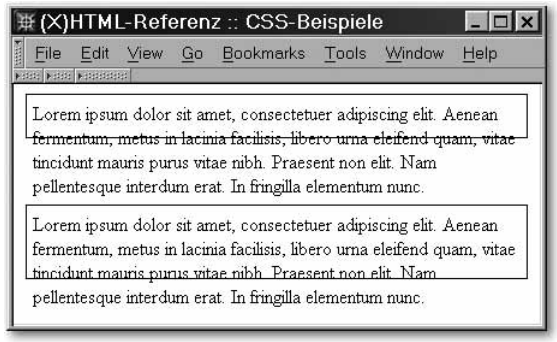

Abbildung 19.39 Bei zu geringer Höhe ragt der Inhalt aus dem Inhaltsbereich.

width

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[laenge], [prozent], auto, inherit*
Defaultwert	*auto*

Das `width`-Property bestimmt die Breite eines Elements. Die hier übergebene Abmessung bezieht sich analog zum `height`-Property ebenfalls auf den inneren Bereich der Box. Ein Margin erhöht die Gesamtbreite entsprechend.

- `width: auto` (Default)
  setzt die Innenbreite auf den durch die Breite des Inhalts erzwungenen Wert

bzw. die Gesamtbreite auf die durch das Elternelement zur Verfügung gestellte Breite und bricht Textinhalt entsprechend um.

- `width: [laenge]`
  setzt die Innenbreite auf den übergebenen Wert und bricht Textinhalt entsprechend um. Ist die Höhe nicht mit `"auto"` festgelegt und der Inhalt höher als die zur Verfügung gestellte Höhe, so ragt er nach unten aus dem Containerbereich heraus (sofern nicht `overflow:hidden` angegeben ist).
- `width: [prozent]`
  setzt die Innenbreite auf den übergebenen Prozentwert, der sich an der Breite des durch das Elternelement zur Verfügung gestellten Raumes misst, und bricht Textinhalt entsprechend um.

Beispiel: **[zB]**

```
<p style="width:200px; height:50px; border-style:solid;
 border-width:1px;border-color:black;">Lorem ipsum ...</p>
```

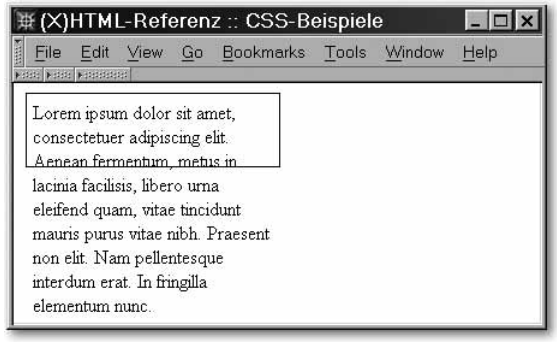

Abbildung 19.40 Container mit festgelegter Breite und Höhe (Mozilla)

max-height

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[laenge], [prozent], none, inherit*
Defaultwert	*none*

Das `max-height`-Property bestimmt die maximale Höhe, die der Innenbereich des Containers dynamisch in Abhängigkeit vom zur Verfügung stehenden Raum oder der Höhe der Inhalte annehmen darf. Es kann eine Längenangabe überge-

ben werden oder ein Prozentwert, der sich auf die Höhe des Elternelements bezieht.

[zB] *Beispiel:* Die Inhalte der Textabsätze konstanter Breite des folgenden Beispiels erfordern unterschiedliche Containerhöhen, die aber bei 50 px durch `max-height` »gedeckelt« werden:

```
<p style="width:200px; max-height:50px;
 border:solid 1px black;">Lorem ipsum.</p>

<p style="width:200px; max-height:50px;
 border:solid 1px black;">Lorem ipsum dolor sit amet,
 consectetuer adipiscing elit.</p>

<p style="width:200px; max-height:50px;
 border:solid 1px black;">Lorem ipsum dolor sit amet,
 consectetuer adipiscing elit. Aenean fermentum, metus in
 lacinia facilisis.</p>
```

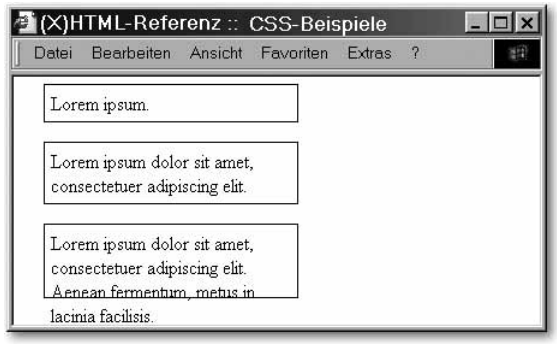

Abbildung 19.41 Container mit variablem Inhalt und gleicher »max-height«

max-width

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[laenge], [prozent], none, inherit*
Defaultwert	–

Das `max-width`-Property bestimmt die maximale Breite, die der Innenbereich des Containers dynamisch in Abhängigkeit vom zur Verfügung stehenden Raum annehmen darf. Es kann eine Längenangabe übergeben werden oder ein Prozentwert, der sich auf die Breite des Elternelements bezieht.

min-height

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[laenge], [prozent], inherit*
Defaultwert	*0*

Das `min-height`-Property bestimmt die minimale Höhe, die der Innenbereich des Containers nicht unterschreiten darf. Es kann eine Längenangabe übergeben werden oder ein Prozentwert, der sich auf die Höhe des Elternelements bezieht.

min-width

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[laenge], [prozent], inherit*
Defaultwert	–

Das `min-width`-Property bestimmt die minimale Breite, die der Innenbereich des Containers nicht unterschreiten darf. Es kann eine Längenangabe übergeben werden oder ein Prozentwert, der sich auf die Breite des Elternelements bezieht.

19.5.5 Kontur-Properties

outline

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	[outline-color] [outline-style] [outline-width]
Defaultwert	*(Defaultwerte der Sub-Properties)*

Das `outline`-Property zeichnet einen Rahmen um den Inhaltsbereich eines Containers, also innerhalb des Außenabstandes. Die Umrisslinie trägt daher, im Gegensatz zum Rahmen, wie er mit `border` erzeugt wird, nicht zu den Abmessungen bei. Outlines können auch Inlineelementen zugewiesen werden.

[zB] *Beispiel:*

```
<p style="padding:5px; margin:15px; border:solid 3px black;">Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
fermentum, metus in lacinia facilisis.</p>

<p style="padding:5px; margin:15px; outline:black 3px solid;">Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
fermentum, metus in lacinia facilisis.</p>
```

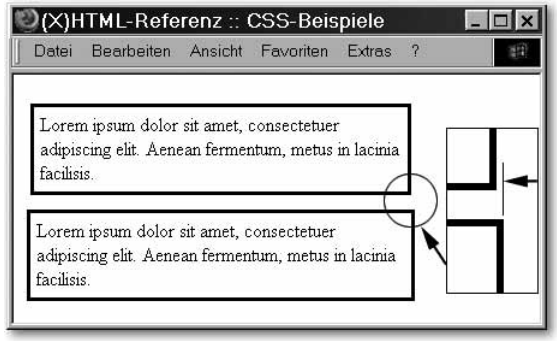

Abbildung 19.42 Die Outline wird außerhalb des Inhaltsbereiches platziert.

outline-color

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[farbname], [farbwert], invert, inherit*
Defaultwert	*invert*

Das `outline-color`-Property legt die Farbe(n) der Umrisslinie oder einer ihrer Seiten fest. Es können in Form von Farbnamen und Farbwerten in Hexadezimal- oder RGB-Schreibweise zwischen einem und vier Werten übergeben werden.

- `outline-color: [wert]`
  Wird ein Wert übergeben, so bezieht sich dieser auf alle vier Seiten der Umrisslinie und weist allen den gleichen Wert zu.
- `outline-color: [wert] [wert]`
  Werden zwei Werte übergeben, so bezieht sich der erste auf die obere und untere Seite der Umrisslinie, der zweite auf diejenige links und rechts.

- `outline-color: [wert] [wert] [wert]`
  Die Übergabe von drei Werten legt mit dem ersten Wert die obere, mit dem zweiten die linke und rechte, sowie mit dem dritten die untere Farbe fest.
- `outline-color: [wert] [wert] [wert] [wert]`
  Vier übergebene Werte legen, in dieser Reihenfolge, die obere, die rechte, die untere und die linke Umrisslinienfarbe fest.

Beispiel: **[zB]**

```
<p style="outline-color:#555555; outline-width:1px; outline-
style:solid;">Lorem ipsum ...</p>

<p style="outline-color:#999999; outline-width:3px; outline-
style:solid;">Lorem ipsum ...</p>

<p style="outline-color:#aaaaaa; outline-width:4px; outline-
style:solid;">Lorem ipsum ...</p>
```

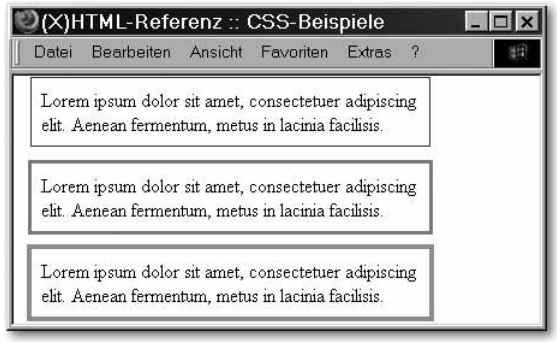

Abbildung 19.43 Umrisslinien in verschiedener Farbe und Breite

outline-style

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*none*, *dotted*, *dashed*, *solid*, *double*, *groove*, *ridge*, *inset*, *outset*, *inherit*
Defaultwert	*none*

Das `outline-style`-Property bestimmt das Aussehen der Umrisslinie. Aus einer Reihe vordefinierter Schlüsselwörter können bis zu vier Werte übergeben werden.

Ein einzelner Wert für `outline-style` bestimmt den Stil der Konturlinie aller vier Seiten, zwei Werte den Stil oben/unten sowie rechts/links, drei Werte den Stil oben, rechts/links sowie unten, vier Werte den Stil aller vier Seiten von oben an im Uhrzeigersinn gerechnet.

- `outline-style: none`
  keine Umrisslinie
- `outline-style: dotted`
  eine gepunktete Umrisslinie
- `outline-style: dashed`
  eine gestrichelte Umrisslinie
- `outline-style: solid`
  es wird eine durchgezogene, einfarbige Umrisslinie gezeigt.
- `outline-style: double`
  eine doppelte, je Seite einfarbige Umrisslinie
- `outline-style: groove`
  eine mehrfarbige Umrisslinie mit plastischem 3D-Effekt (vertiefte Umrisslinie)
- `outline-style: ridge`
  eine invertierte mehrfarbige Umrisslinie mit plastischem 3D-Effekt (erhabene Umrisslinie)
- `outline-style: inset`
  ein zweifarbiger Umriss mit Beleuchtungseffekt oben links
- `outline-style: outset`
  ein zweifarbiger Umriss mit Beleuchtungseffekt unten rechts

[zB] *Beispiel:*

Das `outline`-Property kann auch auf Inlineelemente angewendet werden:

```
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. <i
style="outline-color:black; outline-width:1px; outline-
style:dashed;">Aenean fermentum, metus in lacinia facilisis, libero
urna eleifend quam,</i> vitae tincidunt mauris purus vitae
nibh...</p>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. <i
style="outline-color:#999999; outline-width:4px; outline-
style:ridge;">Aenean fermentum, metus in lacinia facilisis, libero
urna eleifend quam,</i> vitae tincidunt mauris purus vitae
nibh...</p>
```

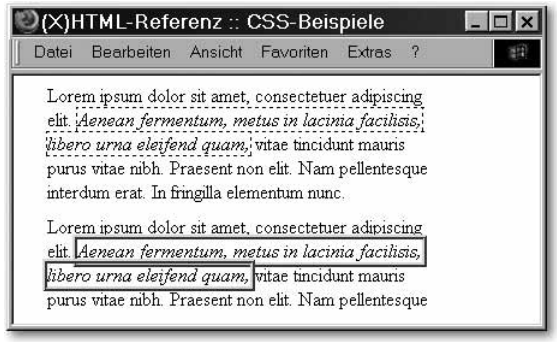

Abbildung 19.44 Outlines mit mit Style »dashed« und »ridge«

outline-width

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[laenge], thin, medium, thick, inherit*
Defaultwert	*medium*

Das `outline-width`-Property bestimmt die Breite der Umrisslinie. Es können bis zu vier Werte in Form von Längenangaben oder dreier vordefinierter Schlüsselwörter in beliebiger Kombination übergeben werden.

Folgende Schlüsselworte können zur Breitenangabe verwendet werden (die Schlüsselworte können auch auf Einzelbreiten angewendet werden):

- `outline-width: thin`
  zeichnet eine dünne Umrisslinie (entspricht 1 px Breite).
- `outline-width: medium` (Default)
  zeichnet eine mitteldicke Umrisslinie (entspricht 3 px Breite).
- `outline-width: thick`
  zeichnet eine dicke Umrisslinie (entspricht 5 px Breite).

Beispiel: **[zB]**

```
<p style="outline-width:thin; outline-style:solid;">Lorem ipsum
...</p>

<p style=" outline-width:medium; outline-style:solid;">Lorem ipsum
...</p>
```

```
<p style=" outline-width:thick; outline-style:solid;">Lorem ipsum
...</p>
```

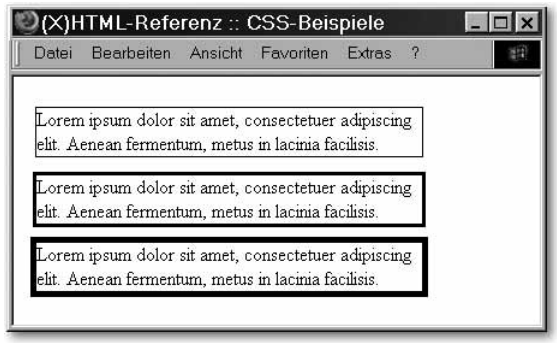

Abbildung 19.45 Outlines der Breite »thin«, »medium« und »thick«

19.5.6 Positioning-Properties

Die Gruppe der Positioning-Properties fasst alle CSS-Eigenschaften zusammen, die die Positionierung von Elementcontainern und weitere, damit zusammenhängende Aspekte betreffen.

bottom

CSS-Level	CSS 2
Anwendung	positionierte Elemente
Vererbung	nein
Werte	*[laenge], [prozent], auto, inherit*
Defaultwert	*auto*

Die Wirkung des `bottom`-Properties ist abhängig sowohl von der `position`-Eigenschaft des positionierten Containers als auch der seines Elterncontainers.

- Bei **relativer Positionierung** bestimmt das `bottom`-Property den Abstand des unteren Randes des positionierten Containers von der Position ohne erfolgte Positionierung. Das `bottom`-Property entspricht in seiner Wirkung somit dem `top`-Property, nur mit umgekehrtem Vorzeichen.

  Das positionierte Element selbst benötigt `position:relative`. Für das Elternelement muss keine entsprechende Angabe gemacht werden.
- Bei **absoluter Positionierung** bestimmt das `bottom`-Property den Abstand zwischen dem unteren Rand des positionierten Elementcontainers und dem unteren Rand des übergeordneten Containers.

Sowohl das Elternelement als auch der in ihm positionierte Container müssen auf `position:absolute` gesetzt sein (Ausnahme: beim Elternelement handelt es sich um das `<body>`-Element).

Es können beliebige (positive oder negative) Längenangaben übergeben werden oder Prozentwerte, die sich auf die Höhe des Elternelements beziehen.

Beispiel: Innerer Container relativ positioniert (siehe Abbildung 19.46 oben; für den äußeren Container braucht kein Positionierungstyp übergeben werden): **[zB]**

```
<div style="border:thin solid; height:85px;">

   <div style="border:thin solid; width:80%;
    position:relative; bottom:10px; left:10px">
        Lorem ipsum ...
   </div>

</div>
```

Äußerer *und* innerer Container absolut positioniert (es brauchen keine expliziten Positionskoordinaten für den äußeren Container übergeben werden):

```
<div style="border:thin solid; height:85px;
 position:absolute;">

   <div style="border:thin solid; width:80%;
    position:absolute; bottom:10px; left:10px">
        Lorem ipsum ...
   </div>

</div>
```

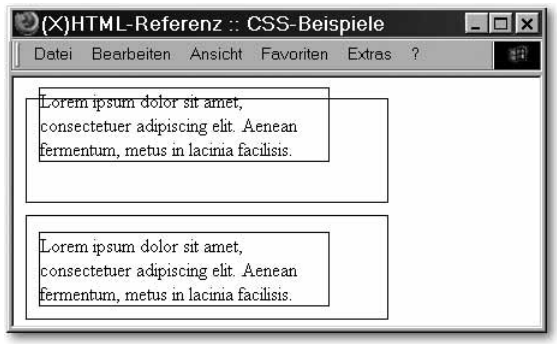

Abbildung 19.46 bottom im relativen (oben) und absolutem Positionskontext

clear

CSS-Level	CSS 2
Anwendung	Blockelemente
Vererbung	nein
Werte	*none, left, right, both, inherit*
Defaultwert	*none*

Das `clear`-Property steht in Zusammenhang mit gefloateten Elementen. Es beendet für den formatierten Elementcontainer das Umfließen eines vorhergehenden, mittels des `float`-Properties nach links oder rechts gefloateten Containers zur angegebenen Seite hin.

- `clear: none` (Default)
  Das Umfließen des Vorgängercontainers wird nicht unterbrochen.
- `clear: left`
  Das Rechtsumfließen eines mit `float:left` formatierten Vorgängercontainers wird unterbrochen; der Container bricht nach der nächsten freien Position am linken Rand des Elternelements um. Mit `float:right` formatierte Vorgänger werden nach wie vor links umflossen.
- `clear: right`
  Das Linksumfließen eines mit `float:right` formatierten Vorgängercontainers wird unterbrochen; der Container bricht nach der nächsten freien Position am rechten Rand des Elternelements um. Mit `float:left` formatierte Vorgänger werden nach wie vor rechts umflossen.
- `clear: both`
  Das Rechts- oder Linksumfließen eines mit `float:left` oder `float:right` formatierten Vorgängercontainers wird unterbrochen; der Container bricht nach der entsprechend nächsten freien Position am linken oder rechten Rand des Elternelements um.

  Anmerkung: Das analoge (jedoch missbilligte) `clear`-Attribut von `<br>` besitzt den Wert `"all"`.

[zB] *Beispiel:* Ohne `clear`-Property umfließen beide folgenden `<div>`-Container das gefloatete Element:

```
<!-- gefloateter Container: -->
<div style="border:thin solid;background-color:#aaaaaa;
height:50px;width:50px;float:left;"></div>
```

```
<!-- umfließt Vorgänger rechts: -->
<div style="border:thin solid;margin-bottom:10px;">
 Lorem ipsum... </div>

<!-- umfließt weiterhin rechts: -->
<div style="border:thin solid;margin-bottom:10px;">
 Lorem ipsum ...</div>
```

Der zweite `<div>`-Container besitzt ein `clear`-Property mit Wert `"left"`. Er bricht demzufolge nach dem nächsten freien linken Rand um:

```
<!-- gefloateter Container: -->
<div style="border:thin solid;background-color:#aaaaaa;
height:50px;width:50px;float:left;"></div>

<!-- umfließt Vorgänger rechts: -->
<div style="border:thin solid;margin-bottom:10px;">
 Lorem ipsum ... </div>

<!-- Umfließen gestoppt: -->
<div style="border:thin solid;clear:left;">
 Lorem ipsum ... </div>
```

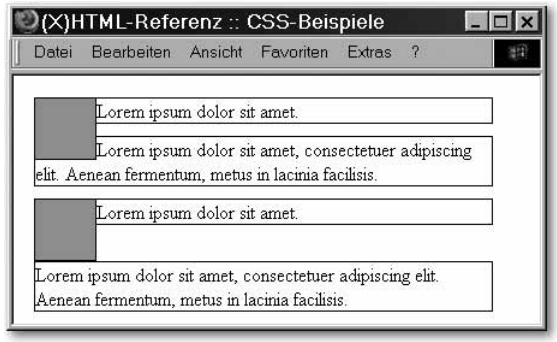

Abbildung 19.47 Der unterste Container rückt dank clear:left an den Rand.

clip

CSS-Level	CSS 2
Anwendung	positionierte Elemente
Vererbung	nein
Werte	*rect([y-top x-re y-bot x-li]), auto, inherit*
Defaultwert	*auto*

Das `clip`-Property beschneidet die Ansicht eines absolut positionierten Containers, ohne dessen Abmessungen dabei zu verändern. Es können ausschließlich rechteckige Ausschnitte festgelegt werden.

- `clip:rect(links, rechts, unten, oben)`
  bestimmt die Position des Ausschnittes durch Festlegung der Ausschnittränder links, rechts, unten und oben ausgehend von der linken und oberen Objektkante. Es können jeweils positive oder negative Werte übergeben werden.

[zB] *Beispiel:* Ein absolut positionierter `<div>`-Container wird beschnitten (d.h. partiell unsichtbar gemacht). Die eigentliche Position des Containers bleibt davon unbeeinflusst. Der erste `clip`-Wert beschriebt die Position des linken Randes der sichtbaren Fläche ab der linken Kante des Containers, der zweite Wert den des rechten Randes – ebenfalls von der linken Kante an gerechnet. Wert drei und viert beschreiben ab der oberen Kante den unteren und den oberen Rand des sichtbaren Bereiches. In Abbildung 19.48 ist die beschnittene Containeransicht (dunkelgrau) mit der unbeschnittenen Ansicht (hellgrau) hinterlegt.

```
<div style="position:absolute;background:#dddddd; width:250px;
clip:rect(10px, 150px, 110px, 10px);">Lorem ipsum...</div>
```

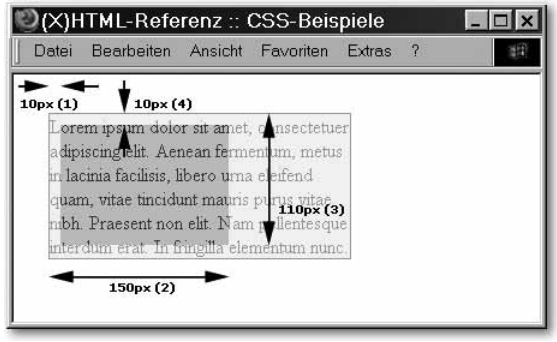

Abbildung 19.48 Beschnittene Ansicht (dunkelgrau) vor Normalansicht

float

CSS-Level	CSS 2
Anwendung	alle nicht-positionierten Elemente
Vererbung	nein
Werte	*left, right, none, inherit*
Defaultwert	*none*

Das `float`-Property bewirkt das Umfließen eines Elements durch Folgeelemente und nimmt das gefloatete Objekt dabei aus dem Zeilenkontext. Das gefloatete Element wird auf seiner gesamten Höhe von Folgeinhalten umflossen, sofern keiner der Folgecontainer vor Ende des Floatobjekts ein `clear`-Property nach der entsprechenden Seite besitzt.

- `float:none` (Default)
  Das Element wird nicht gefloatet.
- `float:left`
  Das Element wird nach links gefloatet und von folgenden Inhalten auf der rechten Seite umflossen.
- `float:right`
  Das Element wird nach rechts gefloatet und von folgenden Inhalten auf der linken Seite umflossen.

Beispiel: **[zB]**

```
<div style="border:thin solid;background-color:#aaaaaa;
 height:50px;width:50px;float:left;"></div>

<div style="border:thin solid;background-color:#aaaaaa;
 height:50px;width:50px;float:right;"></div>

<div style="margin-bottom:10px;">Lorem ipsum ...</div>

<div style="margin-bottom:10px;">Lorem ipsum ...</div>
```

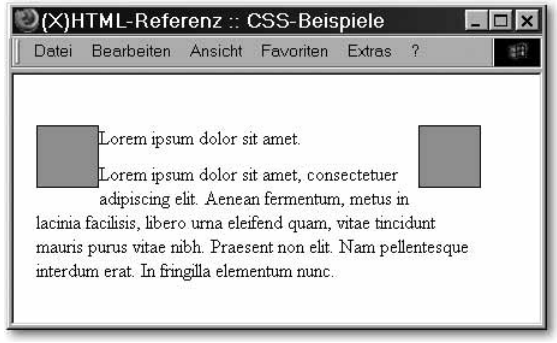

Abbildung 19.49 Links und rechts gefloatete Container mit Folgeelementen

Für das unmittelbar übergeordnete Element ist ein Float-Objekt quasi nicht vorhanden, was seine Dimensionierung betrifft – der Zeilenkontext richtet sich in der Höhe einzig nach den nicht gefloateten Inhalten. Dies kann dazu führen, dass gefloatete Objekte über die Grenzen des sie eigentlich enthaltenden Objekts hinausragen.

Tipp

Eine Lösung hierfür besteht darin, das *übergeordnete* Objekt seinerseits zu floaten (siehe Abbildung 19.50).

[zB] *Beispiel:* Zwei gefloatete Objekte in einem *nicht-gefloateten* Container:

```
<div style="border:thin solid;background-color:#dddddd;
 padding:5px;">

   <div style="border:thin solid;
    background-color:#aaaaaa; height:50px;
    width:50px;float:left;"></div>

   <div style="border:thin solid;
    background-color:#aaaaaa;
height:50px;width:50px;float:right;"></div>

 Lorem? </div>
```

Zwei gefloatete Objekte in einem *gefloateten* Container:

```
<div style="border:thin solid;background-color:#dddddd;
 clear:both; padding:5px;float:left;">

   <div style="border:thin solid;
    background-color:#aaaaaa; height:50px;
    width:50px;float:left;"></div>

   <div style="border:thin solid;
    background-color:#aaaaaa;
height:50px;width:50px;float:right;"></div>

 Lorem! </div>
```

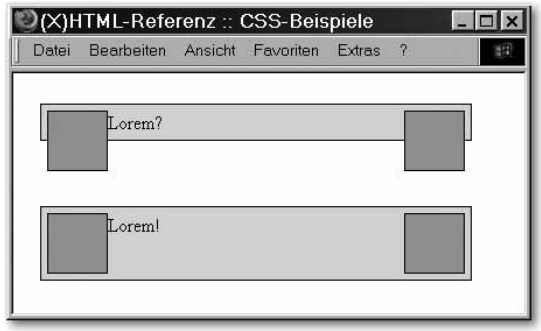

Abbildung 19.50 Nicht-gefloateter und gefloateter (unten) Umfassungscontainer

left

CSS-Level	CSS 2
Anwendung	positionierte Elemente
Vererbung	nein
Werte	*[laenge], [prozent], auto, inherit*
Defaultwert	*auto*

Die Wirkung des `left`-Properties ist abhängig sowohl von der `position`-Eigenschaft des positionierten Containers als auch der seines Elterncontainers.

- Bei **relativer Positionierung** bestimmt das `left`-Property den Abstand des linken Randes des positionierten Containers von der Position ohne erfolgte Positionierung. Das `left`-Property entspricht in seiner Wirkung somit dem `right`-Property, nur mit umgekehrtem Vorzeichen.

  Das positionierte Element selbst benötigt `position:relative`. Für das Elternelement muss keine entsprechende Angabe gemacht werden.
- Bei **absoluter Positionierung** bestimmt das `left`-Property den Abstand zwischen dem linken Rand des positionierten Elementcontainers und dem linken Rand des übergeordneten Containers.

  Sowohl das Elternelement als auch der in ihm positionierte Container müssen auf `position:absolute` gesetzt sein. Ansonsten orientiert sich das positionierte Element am linken Rand des `<body>`-Elements.

Es können beliebige (positive oder negative) Längenangaben übergeben werden oder Prozentwerte, die sich auf die Breite des Elternelements beziehen.

Beispiel: Äußerer *und* innerer Container *absolut* positioniert (siehe Abbildung 19.51 oben; es brauchen keine expliziten Positionskoordinaten für den äußeren Container übergeben werden): **[zB]**

```
<div style="border:thin solid; background-color:#cccccc;
 width:350px; height:85px;position:absolute;">

   <div style="border:thin solid;background-color:#ffffff;
    width:250px; height:70px; position:absolute;
    left:-10px;">Lorem ipsum ...</div>

</div>
```

Innerer Container *relativ* positioniert (für den äußeren Container braucht kein Positionierungstyp übergeben werden):

```
<div style="border:thin solid; background-color:#cccccc;
 width:350px; height:85px;">

   <div style="border:thin solid; background-color:#ffffff;
    width:250px; height:70px; position:relative;
left:30px;">Lorem ipsum ...</div>

</div>
```

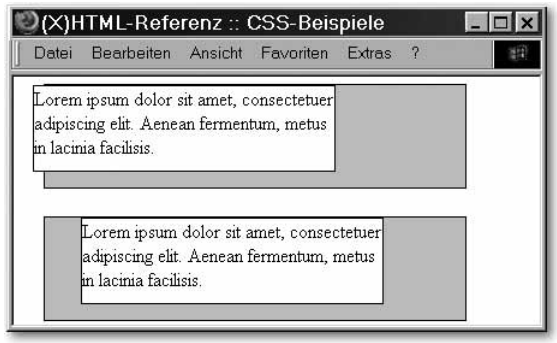

Abbildung 19.51 »left« im absoluten (oben) und relativen Positionskontext

overflow

CSS-Level	CSS 2
Anwendung	Blockelemente, Tabellenzellen, `display:table-cell`
Vererbung	nein
Werte	*visible, hidden, scroll ,auto, inherit*
Defaultwert	*visible*

Das `overflow`-Property legt das Verhalten dimensionierter Container ihrem Inhalt gegenüber fest, falls dessen Abmessungen die vorgesehenen Innenabmessungen des Elements sprengen.

- `overflow: visible` (Default)
  Der überschüssige Inhalt ist sichtbar, ragt dabei jedoch (browserabhängig) über die Grenzen des Containers hinaus.

[»] *Hinweis:* Die Positionierung eines Folgeelements im Flow richtet sich nach der für den Container angegebenen Größe, auch wenn es dadurch zu einer Überlagerung durch den Inhalt des Vorgängers kommt.

- `overflow: hidden`
  Der überschüssige Inhalt wird abgeschnitten, so dass die definierten Abmessungen des Containers eingehalten werden können.
- `overflow: scroll`
  Der Container wird mit den angegebenen Abmessungen dargestellt, aber mit horizontalen *und* vertikalen Scrollbalken versehen, die es ermöglichen, die nicht sichtbaren Teile des Inhaltes zu erreichen.

  Hinweis: Die Scrollbalken werden auch dann angezeigt, wenn die Inhaltsmenge dies nicht erfordert – die im Containerinneren verfügbare Fläche ist dadurch in jedem Fall reduziert. **[«]**
- `overflow: auto`
  Der Container wird mit den angegebenen Abmessungen dargestellt und jeweils an der Seite mit Scrollbalken versehen, für welche die Inhalte die Innenabmessungen übersteigen. Enthält der Container nur Text, der umbrochen werden kann, wird nur der vertikale Scrollbalken eingesetzt. In der Regel wird man `"auto"` daher dem Wert `"scroll"` vorziehen.

Beispiel: **[zB]**

```
<div style="border:thin solid; background-color:#ffffff;
width:300px; height:73px; overflow:hidden;">Lorem ipsum ...
</div>

<div style="border:thin solid; background-color:#ffffff;
width:300px; height:73px; overflow:scroll;">Lorem ipsum ...
</div>
```

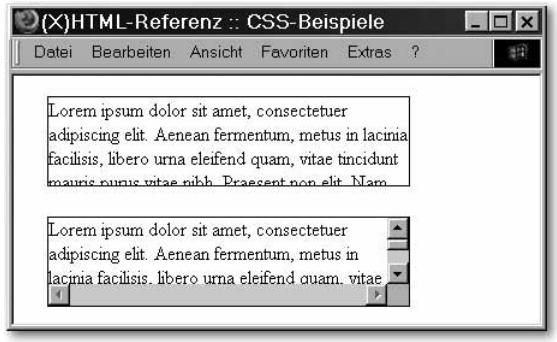

Abbildung 19.52 Container mit overflow:hidden und overflow:scroll

position

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*static, absolute, relative, fixed, inherit*
Defaultwert	*static*

Das `position`-Property bestimmt die Art, in der Positionsangaben bezüglich des positionierten Elements interpretiert werden.

- `position: static` (Default)
  Die Art der Positionierung entspricht der des Elements im normalen Dokumentfluss. Etwaige Angaben zur Positionierung (`left`, `top`, `bottom`, `right`) werden ignoriert.
- `position: relative`
  Die Positionierung bezieht sich auf die Position des Elements im normalen Dokumentfluss. Etwaige Angaben zur Positionierung (`left`, `top`, `bottom`, `right`) beziehen sich auf diese Ursprungsposition.
- `position: absolute`
  Die Positionierung mit `left`, `top`, `right` und `bottom` bezieht sich auf einen absoluten Koordinatenursprung, der entweder durch die obere linke Ecke des Browserfensters (d.i. des `<body>`-Elements) oder durch die obere linke Ecke des übergeordneten Elementcontainers gebildet wird, sofern dieser ebenfalls absolut oder fixiert (`"fixed"`) positioniert ist.
- `position: fixed`
  Der Elementcontainer bleibt beim Scrollen relativ zu seinem Ursprung fixiert. Die Positionierung mit `left`, `top`, `right` und `bottom` bezieht sich auf einen absoluten Koordinatenursprung, der durch die obere linke Ecke des Browserfensters gebildet wird.

[zB] *Beispiel:*

```
<div style="border:thin solid; width:300px; height:100px;">
 Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
 <i style="position:relative; top:5px; left:15px;
  background-color:#cccccc;">Aenean fermentum</i>, metus in
 lacinia facilisis, libero urna eleifend quam, vitae
 tincidunt mauris purus vitae nibh.
</div>
```

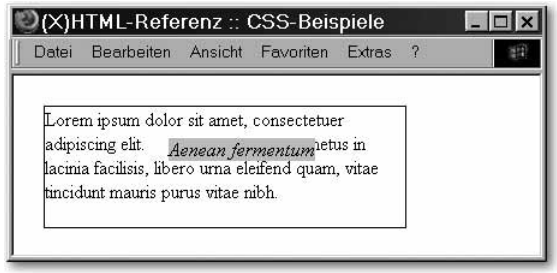

Abbildung 19.53 Relativ positionierter Inlinecontainer in einer Box

right

CSS-Level	CSS 2
Anwendung	positionierte Elemente
Vererbung	nein
Werte	*[laenge], [prozent], auto, inherit*
Defaultwert	*auto*

Die Wirkung des `right`-Properties ist abhängig sowohl von der `position`-Eigenschaft des positionierten Containers als auch der seines Elterncontainers.

- Bei *relativer Positionierung* bestimmt das `right`-Property den *negativen* Abstand des *linken* Randes des positionierten Containers von der Position ohne erfolgte Positionierung. Das `right`-Property entspricht in seiner Wirkung somit dem `left`-Property, nur mit umgekehrtem Vorzeichen.

  Das positionierte Element selbst benötigt `position:relative`. Für das Elternelement muss keine entsprechende Angabe gemacht werden.
- Bei *absoluter Positionierung* bestimmt das `right`-Property den Abstand zwischen dem rechten Rand des positionierten Elementcontainers und dem rechten Rand des übergeordneten Containers.

  Sowohl das Elternelement als auch der in ihm positionierte Container müssen auf `position:absolute` gesetzt sein. Ansonsten orientiert sich das positionierte Element am linken Rand des `<body>`-Elements.

Es können beliebige (positive oder negative) Längenangaben übergeben werden oder Prozentwerte, die sich auf die Breite des Elternelements beziehen.

Beispiel: Innerer Container *relativ* positioniert (siehe Abbildung 19.54; für den äußeren Container braucht kein Positionierungstyp übergeben werden): **[zB]**

```
<div style="border:thin solid; background-color:#cccccc;
 width:300px; height:85px;">

   <div style="border:thin solid; background-color:#ffffff;
    width:250px; height:70px;
    position:relative; right:30px;">Lorem ipsum ....</div>

</div>
```

Äußerer *und* innerer Container *absolut* positioniert (es brauchen keine expliziten Positionskoordinaten für den äußeren Container übergeben werden):

```
<div style="border:thin solid; background-color:#cccccc;
 width:300px; height:85px; position:absolute;">

   <div style="border:thin solid;background-color:#ffffff;
    width:250px; height:70px;
    position:absolute; right:-30px;">Lorem ipsum ...</div>

</div>
```

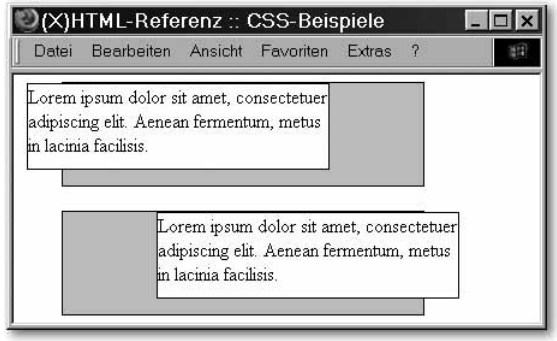

Abbildung 19.54 »right« im relativen (oben) und absoluten Positionskontext

top

CSS-Level	CSS 2
Anwendung	positionierte Elemente
Vererbung	nein
Werte	*[laenge], [prozent], auto, inherit*
Defaultwert	*auto*

Die Wirkung des `top`-Properties ist abhängig sowohl von der `position`-Eigenschaft des positionierten Containers als auch der seines Elterncontainers.

- Bei *relativer Positionierung* bestimmt das `top`-Property den Abstand des oberen Randes des positionierten Containers von der Position ohne erfolgte Positionierung. Das `top`-Property entspricht in seiner Wirkung somit dem `bottom`-Property, nur mit umgekehrtem Vorzeichen.

  Das positionierte Element selbst benötigt `position:relative`. Für das Elternelement muss keine entsprechende Angabe gemacht werden.
- Bei *absoluter Positionierung* bestimmt das `top`-Property den Abstand zwischen dem oberen Rand des positionierten Elementcontainers und dem oberen Rand des übergeordneten Containers.

  Sowohl das Elternelement als auch der in ihm positionierte Container müssen auf `position:absolute` gesetzt sein. Ansonsten orientiert sich das positionierte Element am oberen Rand des `<body>`-Elements.

Es können beliebige (positive oder negative) Längenangaben übergeben werden oder Prozentwerte, die sich auf die Höhe des Elternelements beziehen.

Beispiel: Innerer Container relativ positioniert (siehe Abbildung 19.55; für den äußeren Container braucht kein Positionierungstyp übergeben werden): **[zB]**

```
<div style="border:thin solid; background-color:#cccccc;
 width:350px; height:70px;">

   <div style="border:thin solid; background-color:#ffffff;
    width:250px; height:70px;
    position:relative; top:20px;">Lorem ipsum ...</div>

</div>
```

Äußerer *und* innerer Container *absolut* positioniert (es brauchen keine expliziten Positionskoordinaten für den äußeren Container übergeben werden):

```
<div style="border:thin solid; background-color:#cccccc;
 width:350px; height:70px; position:absolute;">

   <div style="border:thin solid; background-color:#ffffff;
    width:250px; height:70px;
    position:absolute; top:-20px;">Lorem ipsum ...</div>

</div>
```

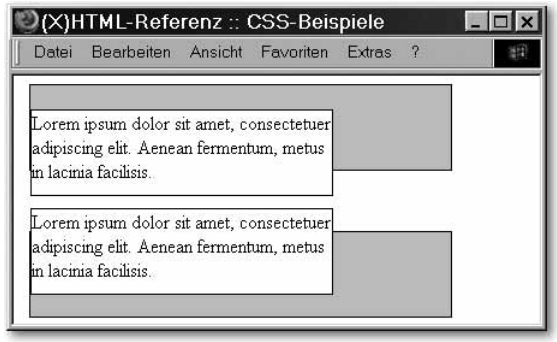

Abbildung 19.55 »top« im relativen (oben) und absoluten Positionskontext

visibility

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*visible, hidden, collapse, inherit*
Defaultwert	*inherit*

Das `visibility`-Property steuert die Sichtbarkeit eines Elementcontainers. Dieser nimmt unabhängig von den dort vorgenommenen Einstellungen stets den gleichen Raum an der ihm angewiesenen Position ein.

- `visibility: visible`
  Das Element ist explizit auf »sichtbar« gesetzt, wird daher also angezeigt.

[»] *Hinweis:* Dies gilt auch dann, wenn das Elternelement auf `"hidden"` gesetzt ist. Die Hintergrundeigenschaften des versteckten Elternelements vererben sich in diesem Falle nicht an das sichtbare Kindelement (auch bei Inlineelementen).

- `visibility: hidden`
  Das Element ist explizit auf »Verstecken« gesetzt, wird daher also verborgen. Hinter dem Element befindliche Inhalte oder Hintergründe werden sichtbar; das Element nimmt jedoch den ihm zugewiesenen Raum ein und beeinflusst so, falls es nicht absolut positioniert ist, den Flow von Nachfolgeelementen.

  Anmerkung: Auf Tabellenelemente angwendet, bewirkt die Einstellung `"hidden"` nur das Verstecken der Zellen*inhalte*, nicht jedoch von Teilen der Tabellenstruktur. Verwenden Sie hierfür `"collapse"`.

- `visibility: collapse`
  Wirkt auf Tabellenelemente wie Zeilen `<tr>` und Spalten `<col>` bzw. Spalten-

gruppen `<colgroup>` und *entfernt* diese aus der Anzeige. Hierbei wird der von ihnen sonst eingenommene Platz jedoch freigemacht, so dass im Flow folgende Elemente nachrücken.

Anmerkung: Auf Nicht-Tabellenelemente angewendet, wirkt die Einstellung `"collapse"` wie `"hidden"`.

- `visibility: inherit` (Default)
  Die Eigenschaft `"inherit"` bindet die Sichtbarkeit eines Elements explizit an die Sichtbarkeit seines Elternelements und blendet dies ein bzw. aus, wenn das Elternelement ein- oder ausgeblendet wird. `"inherit"` ist der Defaultwert für die Sichtbarkeitsangabe; er muss also normalerweise nicht angegeben werden.

Beispiel: Ein unsichtbarer, absolut positionierter Container (in Abbildung 19.56 zur Verdeutlichung ausgegraut mit dargestellt) enthält einen nicht positionierten, explizit »sichtbar« eingestellten Inlinecontainer und einen weiteren absolut positionierten Container, der ebenfalls explizit »sichtbar« gesetzt ist. **[zB]**

```
<div style="border:thin solid; background:#cccccc;
 position:absolute;top:80px;left:80px;visibility:hidden;">
·
·  <span style="visibility:visible;border:1px solid black;">
·   Lorem!</span> top:70px; left:80px;
·
·  <div style="border:thin solid;background:#aaaaaa; ·
position:absolute; visibility:visible; ·   top:90px;left:40px;">
·   Lorem ipsum ... top:90px;left:40px;
·  </div>
·
</div>
```

Abbildung 19.56 Zwei sichtbare Kindcontainer eines unsichtbaren Elements

In zwei nebeneinander angeordneten, ansonsten vom Aufbau identischen Tabellen wird die Mittelzeile der rechten mit `visibility:collapse` versteckt. Ein nachfolgender Inhalt (hier ein Textabsatz) wandert in den so freigemachten Raum ein.

```
<table width="150" frame="box" rules="all"
 style="float:left;margin-right:10px;">
   <tr>
     <td>1</td><td>2</td><td>2</td>
   </tr>
   <tr style="background:#aaaaaa;">
     <td>4</td><td>5</td><td>6</td>
   </tr>
   <tr>
     <td>7</td><td>8</td><td>9</td>
   </tr>
</table>

<table width="150" frame="box" rules="all">
   <tr>
     <td>1</td><td>2</td><td>2</td>
   </tr>
   <tr style="visibility:collapse;">
     <td>4</td><td>5</td><td>6</td>
   </tr>
   <tr>
     <td>7</td><td>8</td><td>9</td>
   </tr>
</table>

<p>Lorem ipsum ...</p>
```

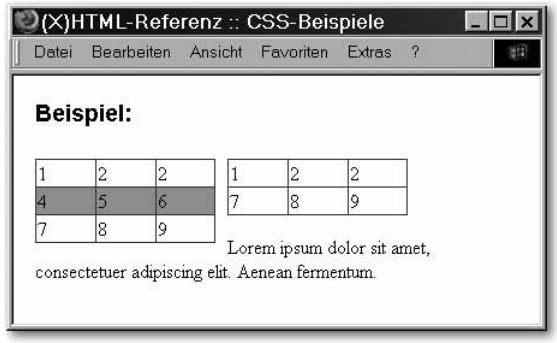

Abbildung 19.57 Tabelle rechts mit ausgeblendeter Mittelzeile

z-index

CSS-Level	CSS 2
Anwendung	positionierte Elemente
Vererbung	nein
Werte	*[numerisch], auto, inherit*
Defaultwert	*auto*

Das `z-index`-Property bestimmt die Tiefenstaffelung sich überlagernder, positionierter Elemente. Es können ganze Zahlen oder der Wert `"auto"` übergeben werden.

- `z-index: auto` (Default)
  Die Überlagerung von hinten nach vorn entspricht der Folge der Elemente in Quelltextreihenfolge.
- `z-index: [numerisch]`
  Es können beliebige ganze Zahlen übergeben werden. Das Element mit dem höchsten `z-index` steht in der Tiefenstaffelung am weitesten vorne. Wird an zwei Elemente der gleiche `z-index` vergeben, so entscheidet die Quelltextreihenfolge über die Tiefenstaffelung. Die Abstände zwischen den vergebenen Werten können beliebig sein – hier ist es weder notwendig noch sinnvoll, eine lückenlose, fortlaufende Reihe zu definieren.[6]

  Achtung: Mozilla und Firefox betrachten Elemente mit `z-index` kleiner 0 als *hinter* dem Dokumenthintergrund liegend, stellen sie also nicht dar. Aus diesem Grunde sollte auf negative `z-index`-Werte verzichtet werden. **[!]**

Beispiel: Die Tiefenstaffelung von sich überlappenden positionierten Ebenen erfolgt in Quelltextreihenfolge von hinten nach vorne, sofern nicht ein `z-index` vergeben wurde (in Abbildung 19.58 die drei rechts liegenden Ebenen). **[zB]**

```
<!-- drei Ebenen ohne z-index: -->

<div style="border:thin solid;background:#aaaaaa;
 height:50px;width:50px;padding:5px;
 position:absolute;left:40px;top:70px;">
   Lorem?</div>

<div style="border:thin solid;background:#cccccc;
 height:50px;width:50px;padding:5px;
```

6 Oft tritt der Fall ein, dass ein weiteres positioniertes Element in die Tiefenstaffelung eingegliedert werden muss. Großzügige Lücken in der `z-index`-Vergabe vereinfachen dies. Gegen höhere `z-index`-Werte wie 100, 200, 300 ist nichts einzuwenden.

```
 position:absolute;left:95px;top:100px;">
   Lorem!</div>

<div style="border:thin solid;background:#eeeeee;
 height:50px;width:50px;padding:5px;
 position:absolute;left:50px;top:130px;">
   Ipsum...</div>

<!-- drei Ebenen mit z-index: -->

<div style="border:thin solid;background:#aaaaaa;
 height:50px;width:50px;padding:5px;
 position:absolute;left:175px;top:70px;z-index:300;">
   Lirum?</div>

<div style="border:thin solid;background:#cccccc;
 height:50px;width:50px;padding:5px;
 position:absolute;left:230px;top:100px;z-index:200">
   Larum!</div>

<div style="border:thin solid;background:#eeeeee;
 height:50px;width:50px;padding:5px;
 position:absolute;left:185px;top:130px;z-index:100">
   Oops...</div>
```

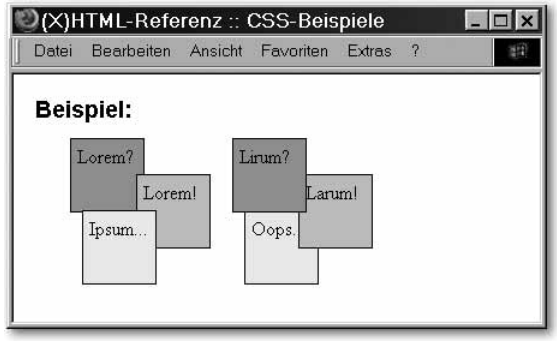

Abbildung 19.58 Drei einander folgende Ebenen ohne (links) und mit z-index

19.5.7 Display-Property

display

CSS-Level	CSS 1; erweitert in CSS 2
Anwendung	global
Vererbung	nein

Werte	*none, inline, block, inline-block, table, inline-table, table-header-group, table-footer-group, table-column-group, table-column, table-row-group, table-row, table-cell, table-caption, list-item, run-in, inherit*
Defaultwert	*inline*

Das `display`-Property legt mit einer großen Anzahl wählbarer Werte die Art und Weise der Darstellung eines Elementcontainers fest. Hierbei können auch grundsätzliche Eigenschaften von HTML-Elementen wie deren Darstellung als Block- oder Inlineelemente überschrieben werden.

Anmerkung: Die Werte `"marker"` und `"compact"`, die in CSS 2.1 abgeschafft wurden, sind hier nicht mit aufgeführt.

- `display: none`
  Die Anzeige des Elements wird vollständig unterdrückt und dieses aus dem Flow genommen. Nachfolgende Inhalte rücken an die frei gewordene Position nach.

Beispiel: Drei Container mit markiertem Inlineelement, das für den zweiten Container auf `visibility:hidden`, für den dritten auf `display:none` gesetzt ist (siehe Abbildung 19.59). Im zweiten Fall beansprucht das unsichtbare Element dennoch Raum, im dritten Fall rücken Folgeinhalte nach. **[zB]**

```
<div style="border:thin solid;margin-bottom:6px;
 padding:5px;"> Lorem ipsum dolor sit amet.
·   <span style="border:dotted 1px;background:#dddddd;">
·    Aenean fermentum.</span>
·
· Libero urna eleifend quam, ...
</div>

<div style="border:thin solid;margin-bottom:6px;
 padding:5px;"> Lorem ipsum dolor sit amet.
·   <span style="visibility:hidden;">
·    Aenean fermentum.</span>
·
· Libero urna eleifend quam, ...
</div>

<div style="border:thin solid;margin-bottom:6px;
 padding:5px;"> Lorem ipsum dolor sit amet.
·   <span style="display:none;">
·    Aenean fermentum.</span>
·
·    Libero urna eleifend quam, ...
</div>
```

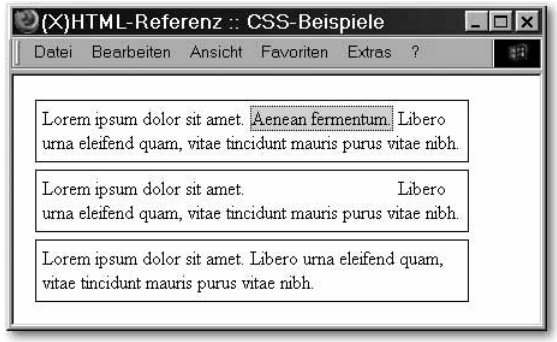

Abbildung 19.59 Eigenschaft visibility:hidden (Mitte) vs. display:none (unten)

- `display: block` (Default für Blockelemente)
  Das Element erhält die Eigenschaften eines Blockelements. Dies bewirkt einen Zeilenumbruch vor und nach den Containergrenzen und die vollständige Anwendbarkeit des CSS-Boxmodells.
- `display: inline` (Default für Inlineelemente)
  Das Element erhält die Eigenschaften eines Inlineelements.
- `display: inline-block`
  Das Element erhält alle Eigenschaften eines Blockelements, jedoch ohne den erzwungenen Zeilenumbruch vor und nach dem Container. Es verhält sich im Zeilenkontext daher ansonsten wie ein Inlineelement, kann aber beispielsweise eine Breite oder ein Margin nach oben oder unten zugewiesen bekommen.

[zB] *Beispiel:* Drei in Blockcontainern eingeschlossene Inlinecontainer erhalten alternativ `display:inline`, `display:block` und `display:inline-block`. Das für alle drei Boxen gesetzte `margin-bottom` wird nur für `display:block` und `display:inline-block` ausgeführt (letzteres nur in Opera 9; siehe Abbildung 19.60):

```
<div style="border:thin solid; margin-bottom:6px;  padding:5px;">
Lorem ipsum dolor sit amet.
·
·   <span style="border:dotted 1px; background:#dddddd;
·    display:inline; margin-bottom:8px; padding:2px;">
·    Aenean fermentum. </span>
·
· Libero urna eleifend quam, ...
</div>

<div style="border:thin solid; margin-bottom:6px;  padding:5px;">
Lorem ipsum dolor sit amet.
```

```
·
·   <span style="border:dotted 1px; background:#dddddd;
·    display:block; margin-bottom:8px; padding:2px;">
·    Aenean fermentum. </span>
·
· Libero urna eleifend quam, ...
</div>

<div style="border:thin solid; margin-bottom:6px;
 padding:5px;"> Lorem ipsum dolor sit amet.
·
·   <span style="border:dotted 1px; background:#dddddd;
·    display:inline-block; margin-bottom:8px; padding:2px;">
·    Aenean fermentum. </span>
·
· Libero urna eleifend quam, ...
</div>
```

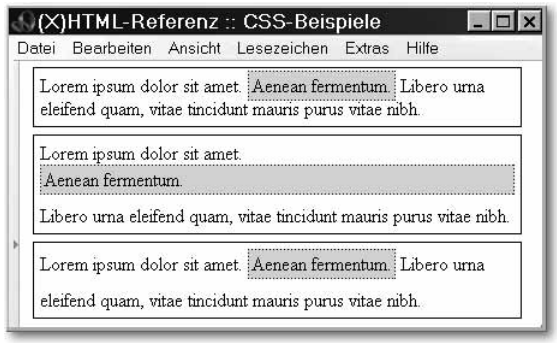

Abbildung 19.60 Display als »inline«, »block« und »inline-block« (Opera 9.02)

- `display: run-in`
  Das Element bricht nach dem vorangehenden Inhalt um, verhält sich jedoch gegenüber seinem Nachfolgerelement, dessen Eigenschaften entsprechend, unterschiedlich: Ist der Nachfolger ein *Blockelement*, so läuft das Run-in-Element als Inlineelement in dessen erste Zeile hinein. Handelt es sich um ein *gefloatetes Element*, so verhält sich das Run-in-Element hingegen wie ein Blockelement und bricht vor den Nachfolger um.

Beispiel: Ein Run-in-Element verursacht nach dem ihm vorangehenden Inhalt stets einen Umbruch, gliedert sich selbst aber in den Inlinecontent eines folgenden Blockelements ein, sofern dieses nicht gefloatet ist. **[zB]**

```
<div> Lorem?
   <span style="display:run-in">
   <b>Dies ist ein Run-In: </b></span>
   <p>Lorem ipsum dolor sit amet ...</p>
</div>

<div> Lorem!
   <span style="display:run-in">
   <b>Dies ist kein Run-In mehr: </b></span>
   <p style="float:left">Ich floate.
    Lorem ipsum dolor sit amet ...</p>
</div>
```

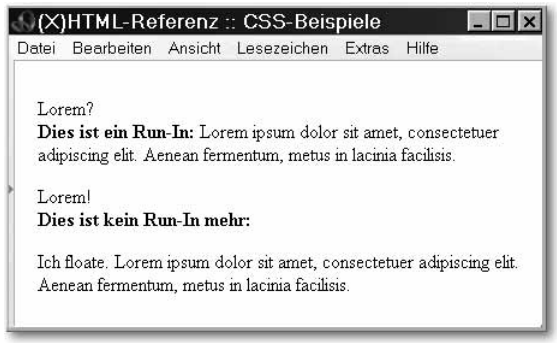

Abbildung 19.61 Run-in-Element vor Block (oben) und Floater (in Opera 9.02)

- `display:list-item`
  Das Element wird als Listenelement dargestellt. Ein übergeordneter Listencontainer muss hierfür nicht definiert werden. Ein so definiertes Listenelement ist grundsätzlich ein Blockelement, wobei die üblichen Defaultverhalten in Bezug auf Bullets gelten, nicht jedoch auf Margins.

[zB] *Beispiel:* Einem beliebigen Element kann auch das Verhalten des `<li>`-Elements zugewiesen werden. Da per Default kein Margin vergeben wird und das Aufzählungselement »outside« positioniert wird, kommt Letzteres auf dem Rand des übergeordneten Elements zu liegen (Abbildung 19.62, oben):

```
<div style="display:list-item;">Rosen</div>
<div style="display:list-item;">Tulpen</div>
<div style="display:list-item;">Nelken</div>
```

Mittels eines Hilfscontainers kann eine Einrückung einfach für den gesamten Listenblock veranlasst werden.

```
<div style="margin-left:30px">
   <div style="display:list-item;">Rosen</div>
   <div style="display:list-item;">Tulpen</div>
   <div style="display:list-item;">Nelken</div>
</div>
```

Abbildung 19.62 CSS-Liste »solo« und mit Einrückung durch Hilfscontainer

Darüber hinaus kann man bei Listenelementen Einfluss nehmen auf `list-style-type` (Bullet oder Zahl – auch eine Grafik kann mittels `list-style-image` über einen URI als Bullet dienen) und auf `list-style-position`. Hier bedeutet `inside` bzw. `outside`, dass der Markerblock (in dem sich das Bullet befindet) innerhalb oder außerhalb des Listenelementblocks sitzt. Für alle drei Eigenschaften gemeinsam existiert auch das Sammelproperty `list-style`.

Darstellung als Tabellenelement

Allen HTML-Elementen können seit CSS 2 über entsprechende `display`-Werte auch die Darstellungseigenschaften von Tabellenelementen zugeordnet werden.

Hier muss darauf geachtet werden, dass mindestens die erforderlichen Rollen definiert werden. Ein Element muss daher die Rolle des `<table>`-Containers übernehmen (`display:table`), ein weiteres die Rolle der Tabellenzeile `<tr>`, bekommt also `display:table-row`. Ein Element schließlich stellt mit `display:table-cell` die Tabellenzelle `<td>` dar.

- `display:table`
  Das Element verhält sich wie eine Tabelle, also als Blockelement.
- `display:inline-table`
  Das Element verhält sich wie eine Tabelle, bleibt dabei jedoch im Inlinekontext. Die Eigenschaft ist ein Pendat zu `block-inline`. Eine Entsprechung wäre die Zuweisung von `display:inline` an ein gewöhnliches `<table>`-Element.

Element	display-Wert
<table>	display:table
<tr>	display:table-row
<td>	display:table-cell
<caption>	display:table-caption
<col>	display:table-column
<colgroup>	display:table-column-group
<tfoot>	display:table-footer-group
<thead>	display:table-header-group
<table> (aber als Inlineelement)	display:inline-table

Tabelle 19.4 Das display-Property für Tabellendarstellung

[zB] *Beispiel:* Mit etwas Mühe kann man eine Tabelle aus anderen HTML-Elementen nachbauen (siehe Abbildung 19.63, oben). Es gibt hierbei jedoch kaum Möglichkeiten, die eine herkömmliche HTML-Tabelle nicht ebenso bieten würde.[7]

Interessant mag die Vorgehensweise dennoch sein, wenn sich die Tabellenstruktur für andere Ausgabegeräte auflösen können soll oder komplett anders dargestellt werden muss. Dies kann durch eine CSS-Umdeutung einer regulären Tabelle im Gegenzug nur schwer zu erreichbar sein.

```
<div style="display:table;width:250px; border:1px dotted;
padding:5px; border-spacing:2px">
    <div style="display:table-row;">
        <span style="display:table-cell; border:1px solid;
padding:3px;">1</span>
        <span style="display:table-cell; border:1px solid;
padding:3px;">2</span>
    </div>
    <div style="display:table-row;">
        <span style="display:table-cell; border:1px solid;
padding:3px;">3</span>
         <span style="display:table-cell;border:1px solid;
padding:3px;">4</span>
    </div>
</div>
```

[zB] *Beispiel:* Die Formatierung eines Containers als **Inline-Tabelle** kann dazu eingesetzt werden, eine Ruby-ähnliche, mehrzeilige Darstellung von Inhalten im Zeilenkontext zu ermöglichen (siehe Abbildung 19.63, unten).

7 Eher weniger, sofern man an die HTML-Attribute denkt, die speziell der Barrierefreiheit von Tabellen dienen.

```
<div style="border:solid 1px; padding:10px;
 line-height:2.2em;"> Das ist eine
<span style="display:inline-table; font-size:70%;
 line-height:60%;vertical-align:middle; text-align:center;">
    <span style="display:table-row">
        <span style="display:table-cell">
         lustige
        </span>
    </span>
    <span style="display:table-row">
        <span style="display:table-cell">
          interessante
        </span>
    </span>
    <span style="display:table-row">
        <span style="display:table-cell">
          spannende
        </span>
    </span>
</span> Ruby-Simulation. Deren Realisierung kann man als
<span style="display:inline-table; font-size:70%;
 line-height:60%;vertical-align:middle; text-align:center;">
    <span style="display:table-row">
        <span style="display:table-cell">
          genau richtig
        </span>
    </span>
    <span style="display:table-row">
        <span style="display:table-cell">
          etwas umst&auml;ndlich
        </span>
    </span>
    <span style="display:table-row">
        <span style="display:table-cell">
          vollkommen absurd
        </span>
    </span>
</span>
 bezeichnen, je nach Laune.</div>
```

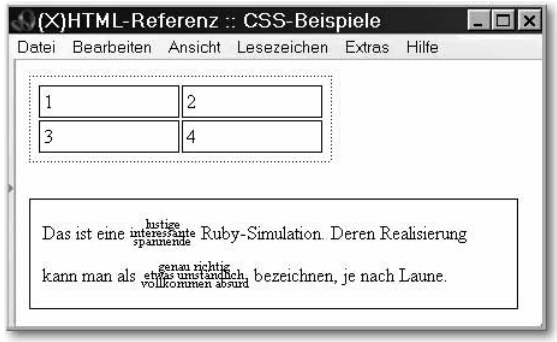

Abbildung 19.63 CSS-Tabelle und Ruby-Simulation in Opera 9.02

19.5.8 Listenstyle-Properties

list-style

CSS-Level	CSS 1
Anwendung	`<ul>`, `<ol>`, `<li>` und Elemente mit `display:list-item`
Vererbung	nein
Werte	*[list-style-type]? [list-style-image]? [list-style-position]?*
Defaultwert	–

Das `list-style`-Property ist ein **Sammelproperty** für die drei grundlegenden Listenelementeigenschaften `list-style-type`, `list-style-image` und `list-style-position`, die hier einzeln oder gemeinsam zugeordnet werden können. Wird ein Einzelproperty nicht übergeben, so gilt dessen Defaultwert.

- `list-style: [list-style-type]`
  Gibt die Art des zu verwendenden Aufzählungselements an. Der Defaultwert ist `"disc"`.
- `list-style: [list-style-image]`
  Gibt den URL einer als Aufzählungselement zu verwendenden Grafik an. Der Defaultwert ist `"none"`.
- `list-style: [list-style-position]`
  Gibt die Position des Aufzählungselements gegenüber der linken Seite der (gedachten) Box um das Listenelement an. Der Defaultwert ist `"outside"`.

Anmerkung: Der Listentyp `list-style-type` und die Listengrafik `list-style-image` können nicht gleichzeitig eingesetzt werden.

[zB] *Beispiel:* Mittels des `list-style`-Properties lassen sich sinnvoll bis zu zwei Listeneigenschaften gleichzeitig definieren:

```
<div style="display:list-item;
 list-style:square">Rosen</div>
<div style="display:list-item;
 list-style:url('pfeil.gif') inside">Tulpen</div>
<div style="display:list-item;
 list-style:decimal-leading-zero">Nelken</div>
<div style="display:list-item;
 list-style:outside">Lorem ipsum ...</div>
<div style="display:list-item;
 list-style:inside">Lorem ipsum ...</div>
<div style="display:list-item;
 list-style:none">Lorem ipsum ...</div>
</div>
```

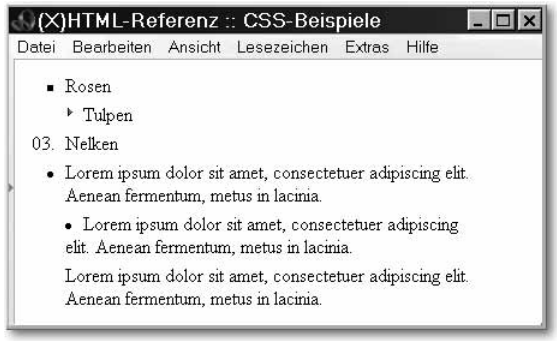

Abbildung 19.64 Verschiedene list-style-Zuweisungen in Opera 9.02

list-style-image

CSS-Level	CSS 1
Anwendung	`<ul>`, `<ol>`, `<li>` und Elemente mit `display:list-item`
Vererbung	nein
Werte	*url([pfad/dateiname]), none, inherit*
Defaultwert	*none*

Das `list-style-image`-Property legt eine Grafik fest, die als Aufzählungezeichen verwendet werden soll. Defaultwert ist `"none"` (keine Grafik) – in diesem Falle tritt das `list-style-type`-Property in Kraft.

Beispiel: **[zB]**

```
<div style="display:list-item;
 list-style-image:url('pfeil.gif')">Tulpen</div>
```

list-style-position

CSS-Level	CSS 1
Anwendung	`<ul>`, `<ol>`, `<li>` und Elemente mit `display:list-item`
Vererbung	Nein
Werte	*inside, outside, inherit*
Defaultwert	*Outside*

Das `list-style-position`-Property bestimmt die Position des Aufzählungselements gegenüber der Box des Listenelements. Defaultwert ist `"outside"`.

- `list-style-position: outside` (Default)
  Das Aufzählungselement wird links außerhalb der Listenbox platziert.
- `list-style-position: inside`
  Das Aufzählungselement wird links innerhalb der Listenbox platziert. Ist der Text des Listenelements mehrzeilig, so umfliesst er das Aufzählungselement.

[zB] *Beispiel:* Die Darstellung ist analog zu der von `list-style:outside` und `list-style:inside` (siehe Abbildung 19.64).

```
<div style="display:list-item;
 list-style-position:outside">Lorem ipsum ...</div>
<div style="display:list-item;
 list-style-position:inside">Lorem ipsum ...</div>
```

list-style-type

CSS-Level	CSS 1
Anwendung	`<ul>`, `<ol>`, `<li>` und Elemente mit `display:list-item`
Vererbung	nein
Werte	*disc, circle, square, decimal, decimal-leading-zero, lower-roman, upper-roman, lower-greek, lower-latin, upper-latin, armenian, georgian, lower-alpha, upper-alpha, none, inherit*
Defaultwert	*disc*

Das `list-style-type`-Property bestimmt die Art des zu verwendenden Aufzählungszeichens. Der Defaultwert ist `"disc"`.

- `list-style-type: disc` (Default)
  Scheibe als Aufzählungszeichen
- `list-style-type: circle`
  Kreis als Aufzählungszeichen

- `list-style-type: square`
  Quadrat als Aufzählungszeichen
- `list-style-type: decimal`
  Dezimalzahlen (1. ,2. , 3. , ...)
- `list-style-type: decimal-leading-zero`
  Dezimalzahlen mit führender Null (01. ,02. , 03. , ...)
- `list-style-type: lower-roman`
  Kleine römische Ziffern (i. ,ii. ,iii. , ...)
- `list-style-type: upper-roman`
  Grosse römische Ziffern (I. ,II. , III., ...)
- `list-style-type: lower-greek`
  Kleine griechische Buchstaben (, , , ...)
- `list-style-type: lower-latin`
  Kleine lateinische Buchstaben (a. ,b., c. , ...)
- `list-style-type: upper-latin`
  Große lateinische Buchstaben (A., B. ,C. , ...)
- `list-style-type: armenian`
  Armenische Nummerierungszeichen
- `list-style-type: georgian`
  Georgische Nummerierungszeichen
- `list-style-type: lower-alpha`
  Kleine ASCII-Zeichen, auch als lateinische Buchstaben bekannt (a. ,b., c. , ...)
- `list-style-type: upper-alpha`
  Große ASCII-Zeichen, auch als lateinische Buchstaben bekannt (A., B. ,C. , ...)
- `list-style-type: none`
  Kein Aufzählungszeichen

Beispiel: Alle Werte von `list-style-type` sind in Abbildung 19.65 in zwei durch Float nebeneinandergelegten Containern demonstriert. (Für georgische und armenische Nummerierung müssen entsprechende Zeichensätze vorliegen.) **[zB]**

```
<!-- linke Spalte: -->
<div style="margin-left:30px; width:180px;float:left">
<div style="5px;display:list-item;
 list-style-type:disc">disc</div>
<div style="display:list-item;
 list-style-type:circle">circle</div>
<div style="display:list-item;
 list-style-type:square">square</div>
```

```
<div style="display:list-item;
 list-style-type:decimal">decimal</div>
<div style="display:list-item;
 list-style-type:decimal-leading-zero">
decimal-leading-zero</div>
<div style="display:list-item;
 list-style-type:lower-roman">lower-roman</div>
<div style="display:list-item;
 list-style-type:upper-roman">upper-roman</div>
<div style="display:list-item;
 list-style-type:lower-greek">lower-greek</div>
</div>

<!-- rechte Spalte: -->
<div style="margin-left:30px; width:150px;">
<div style="display:list-item;
 list-style-type:lower-latin">lower-latin</div>
<div style="display:list-item;
 list-style-type:upper-latin">upper-latin</div>
<div style="display:list-item;
 list-style-type:armenian">armenian</div>
<div style="display:list-item;
 list-style-type:georgian">georgian</div>
<div style="display:list-item;
 list-style-type:lower-alpha">lower-alpha</div>
<div style="display:list-item;
 list-style-type:upper-alpha">upper-alpha</div>
<div style="display:list-item;
 list-style-type:none">none</div>
</div>
```

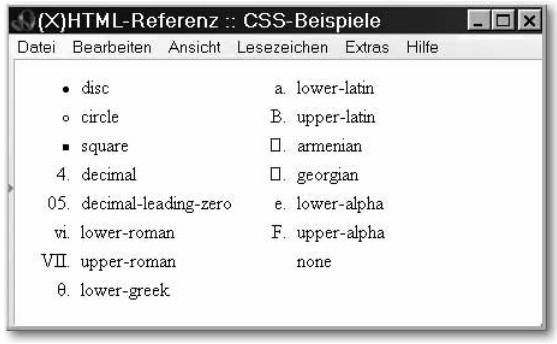

Abbildung 19.65 Die in Opera 9.02 darstellbaren Listentypen

19.5.9 Tabellen-Properties

In Zusammenhang mit »natürlichen« HTML-Tabellenelementen und beliebigen Elementen, die per `display`-Property mit `display:table` oder `display:inline-table` CSS-Tabellenelementeigenschaften zugewiesen bekommen haben, existieren weitere CSS-Eigenschaften, um die Darstellung exakter zu steuern.

border-collapse

CSS-Level	CSS 2
Anwendung	`<table>`, `display:table`-sowie `display:inline-table`
Vererbung	ja
Werte	*separate, collapse, inherit*
Defaultwert	*separate*

Das `border-collapse`-Property bestimmt, ob Rahmen um Tabellenzellen (sofern diese definiert sind) zusammenfallen oder, was der Vorgabe entspricht, durch Abstände separiert dargestellt werden sollen.

- `border-collapse: separate` (Default)
  Die Rahmen benachbarter Zellen fallen nicht zusammen. Der genaue Abstand (entsprechend HTML-`"cellspacing"`) wird mit den `border-spacing`-Property bestimmt. Beträgt der Abstand Null, addieren sich die Rahmenbreiten optisch.
- `border-collapse: collapse`
  Die Rahmen benachbarter Zellen fallen so zusammen, so dass ihre Breiten sich nicht addieren.

Hinweis: Ein für die Tabelle definierter separater Rahmen wird ebenfalls den Zellenrahmen zugeschlagen und Paddingwerte der Tabelle ignoriert. **[«]**

Beispiel: Zwei identische CSS-Tabellen unterschieden sich nur anhand ihres `border-collapse`-Properties. Bei der unteren Tabelle (Abbildung 19.66) entfällt Padding und der äußere Rahmen in der Darstellung, obwohl diese Eigenschaften definiert sind. **[zB]**

```
<div style="display:table; width:250px;
 border:1px dotted; padding:5px; border-collapse:separate">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">2</span>
   </div>
```

```
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">3</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>

<div style="display:table; width:250px;
 border:1px dotted; padding:5px; border-collapse:collapse">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">2</span>
   </div>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">3</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>
```

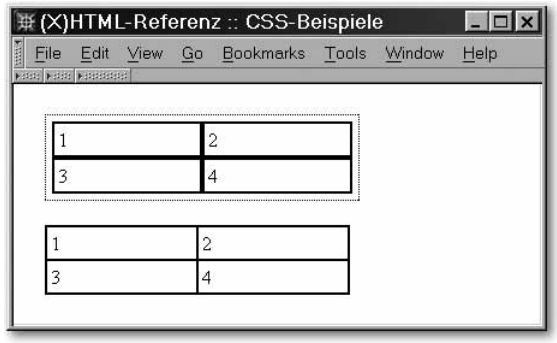

Abbildung 19.66 CSS-Tabellen mit border-collapse »separate« und »collapse«

border-spacing

CSS-Level	CSS 2
Anwendung	`<table>`, `display:table`-sowie `display:inline-table`
Vererbung	ja
Werte	*[laenge], inherit*
Defaultwert	*0*

Das `border-spacing`-Property bestimmt gleich dem `cellspacing`-Attribut in HTML den Abstand zwischen den Rahmenlinien benachbarter Tabellenzellen. Es können beliebige Längenangaben vergeben werden.

Hinweis: Das Property `border-collapse` muss hierfür auf `"separate"` stehen oder nicht gesetzt sein. **[«]**

Beispiel: Zwei identische CSS-Tabellen unterschieden sich nur anhand ihres `border-spacing`-Properties. Bei der unteren Tabelle (Abbildung 19.67) ist `border-collapse` nicht explizit auf `"separate"` gesetzt. (Dies ist möglich, da dies ohnehin der Vorgabe entspricht.) **[zB]**

```
<div style="display:table; width:250px;
 border:1px dotted; padding:5px;  border-collapse:separate;
 border-spacing:1px;">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">2</span>
   </div>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">3</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>

<div style="display:table; width:250px;
 border:1px dotted; padding:5px; border-spacing:5px">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">2</span>
   </div>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">3</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>
```

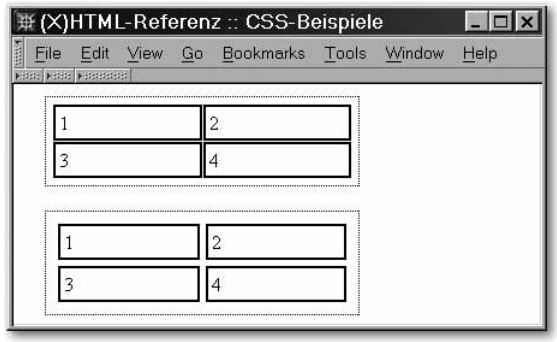

Abbildung 19.67 CSS-Tabellen mit verschiedenen border-spacing-Werten

caption-side

CSS-Level	CSS 2 (geändert in CSS 2.1)
Anwendung	`<caption>`, `display-table`, `display:inline-table` und `display:table-caption`-Elemente
Vererbung	ja
Werte	*top, bottom, inherit*
Defaultwert	*top*

Das `caption-side`-Property legt fest, ob eine Tabellenüberschrift über oder unter der Tabelle erscheinen soll. Da die Eigenschaft sich vererbt, kann sie entweder direkt dem Caption-Element oder aber auch der Tabelle selbst zugewiesen werden.

[»] *Hinweis:* In CSS 2.1 wurden die Werte `"left"` und `"right"` abgeschafft, die eine Caption neben der Tabelle darstellen sollten. Diese Werte sind hier deshalb nicht aufgeführt.

- `caption-side: top` (Default)
  Die Überschrift erscheint über der Tabelle
- `caption-side: bottom`
  Die Überschrift erscheint unter der Tabelle

[zB] *Beispiel:* Für die obere der beiden Tabellen in Abbildung 19.68 ist `caption-side` direkt im Tabellencontainer definiert, für die untere im Caption-Element selbst. (Die Angabe `"top"` für die obere Tabelle könnte auch entfallen, da sie dem Vorgabewert entspricht.)

```
<div style="display:table; width:250px; border:1px dotted;
 padding:5px; border-spacing:2px; caption-side:top;">
  <span style="display:table-caption;">
```

```
   Tabellen&uuml;berschrift</span>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">2</span>
   </div>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">3</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>

<div style="display:table; width:250px;
 border:1px dotted; padding:5px; border-spacing:2px">
  <span style="display:table-caption; caption-side:bottom">
  Tabellen&uuml;berschrift</span>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">2</span>
   </div>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">3</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>
```

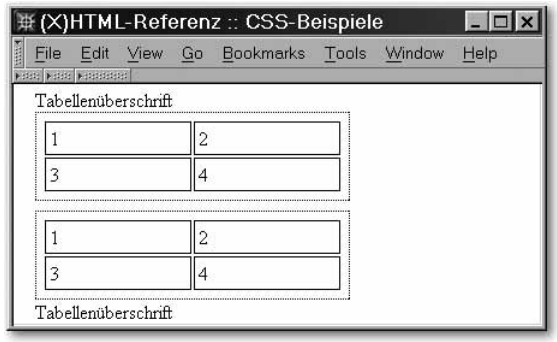

Abbildung 19.68 Die Werte »top« und »bottom« für caption-side

empty-cells

CSS-Level	CSS 2
Anwendung	`<table>`, `display:table`-sowie `display:inline-table`
Vererbung	ja
Werte	*show, hide, inherit*
Defaultwert	*show*

Das `empty-cells`-Property bestimmt, ob für leere Tabellenzellen deren Rahmen, Hintergrundfarbe und -bild angezeigt oder versteckt werden soll.

- `empty-cells: show` (Default)
  Rahmen und Hintergründe leerer Zellen werden angezeigt.
- `empty-cells: hide`
  Leere Zellen werden mitsamt Rahmen und Hintergründen versteckt.

[zB] *Beispiel:* Die obere der beiden Tabelle in Abbildung 19.69 besitzt die Anweisung `empty-cells:show`. In diesem Fall werden die Rahmen der leeren Zellen angezeigt. Die Rahmen der leeren Zellen der unteren Tabellen werden hingegen versteckt, da in ihrem Fall `empty-cells` auf `"hide"` gesetzt ist.

```
<div style="display:table; width:250px; border:1px dotted;
 padding:5px; border-spacing:2px; empty-cells:show;">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;"></span>
   </div>
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;"></span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>

<div style="display:table; width:250px; border:1px dotted;
 padding:5px; border-spacing:2px; empty-cells:hide;">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">1</span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;"></span>
   </div>
   <div style="display:table-row;">
```

```
      <span style="display:table-cell; border:2px solid;
       padding:3px;"></span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">4</span>
   </div>
</div>
```

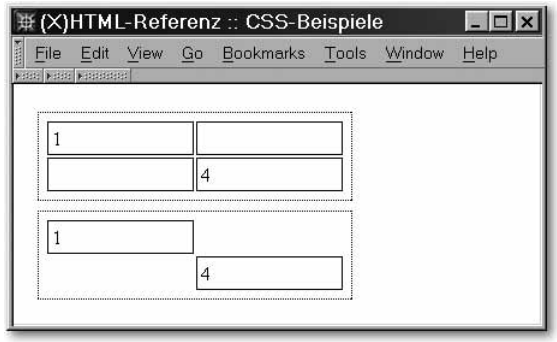

Abbildung 19.69 Leere Zellen mit empty-cells anzeigen oder verstecken

table-layout

CSS-Level	CSS 2
Anwendung	`<table>`, `display:table`-sowie `display:inline-table`
Vererbung	ja
Werte	*auto, fixed, inherit*
Defaultwert	*auto*

Das `table-layout`-Property bestimmt, ob die Abmessungen der Tabelle als Höchstmaße interpretiert werden sollen oder die Tabelle sich den Inhalten in der Breite anpassen darf. Defaultwert ist `"auto"`.

Hinweis: Der Wert `"fixed"` kann zu Anzeigeproblemen führen, wenn die tatsächliche Breite der Inhalte die der Tabelle erlaubten Abmessungen sprengt. Ansonsten wird dem Browser der Renderingvorgang erleichtert, da andernfalls erst die Inhalte eingelesen und interpretiert werden müssten, bevor die Tabelle dargestellt werden kann. **[«]**

- `table-layout: auto` (Default)
  Die resultierende Anzeigebreite der Tabelle ergibt sich über die angegebene Mindestbreite hinaus aus den Abmessungen der Inhalte.

- `table-layout: fixed`
  Die angegebene Tabellenbreite wird als maßgeblich betrachtet, was den Renderingvorgang beschleunigt. Eventuell überstehende Inhalte werden abgeschnitten oder (die Regel) ragen über den Zellrand hinaus.

[zB] *Beispiel:* Die obere Tabelle in Abbildung 19.70 besitzt `table-layout:auto`, wird also normal gerendert, während die untere durch `table-layout:fixed` in ihrer Darstellungsbreite festgelegt ist. Die durch Non-breaking-Spaces verbundenen Textinhalte ragen daher über die Zellränder nach rechts hinaus.

```
<div style="display:table; width:250px; border:1px dotted;
 padding:5px; border-spacing:2px; table-layout:auto;">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">
       Lorem ipsum dolor sit amet.
      </span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">
       Lorem ipsum dolor sit amet.
      </span>
   </div>
   <div style="display:table-row;">
...
   </div>
</div>
```

```
<div style="display:table; width:250px; border:1px dotted;
 padding:5px; border-spacing:2px; table-layout:fixed;">
   <div style="display:table-row;">
      <span style="display:table-cell; border:2px solid;
       padding:3px;">
       Lorem ipsum dolor sit amet.
      </span>
      <span style="display:table-cell; border:2px solid;
       padding:3px;">
       Lorem ipsum dolor sit amet.
      </span>
   </div>
   <div style="display:table-row;">
...
   </div>
</div>
```

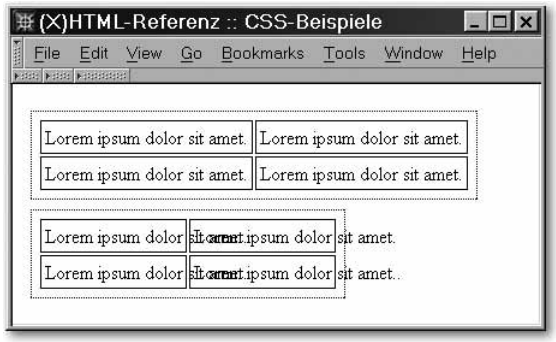

Abbildung 19.70 Tabellenlayout mit table-layout »auto« und »fixed«

19.5.10 Sonstige Properties

CSS kennt einige wenige Eigenschaften, die nicht das Seitenlayout, sondern das Interface beeinflussen. Hierzu gehört das `cursor`-Property.

cursor

CSS-Level	CSS 2
Anwendung	global
Vererbung	ja
Werte	*url(), auto, crosshair, default, pointer, move, e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize, text, wait, help, inherit*
Defaultwert	*auto*

Das `cursor`-Property bestimmt das Aussehen des Mauszeigers beim Überfahren des Elementcontainers mit der Maus. Der Defaultwert ist `"auto"`, d.h. der Cursor nimmt die Form an, die durch das jeweilige Element vorgegeben ist (beispielsweise `"pointer"` für Links oder `"text"` für Eingabefelder).

Die exakten Formen der Mauszeiger sind durch das System und die Nutzereinstellungen vorgegeben und nicht durch CSS-Angaben beeinflussbar (Windows beispielsweise bietet hier rund ein Dutzend Varianten an).

Hinweis: Um Irritationen zu vermeiden, sollte man nicht willkürlich Mauszeigerformen einsetzen, die Möglichkeiten vortäuschen, die das entsprechende Element nicht bietet. Zudem sind bestimmte Cursorformen (`"crosshair"`, `"text"`) wegen der Abwesenheit des Schlagschattens für Sehbehinderte schwerer wahrnehmbar. **[«]**

- `cursor: auto` (Default)
  Mauszeiger entspricht den Systemvorgaben.
- `cursor: default`
  zeigt plattformunabhängig den Standard-Mauszeiger.
- `cursor: crosshair`
  nimmt die Form eines Fadenkreuzes an (kein Schatten).
- `cursor: pointer`
  nimmt die Form einer Hand an; signalisiert Aktivierbarkeit (Link etc.).
- `cursor: move`
  nimmt die Form eines Doppelkreuzes an; signalisiert Bewegbarkeit.
- `cursor: text`
  Textcursor (kein Schatten); signalisiert Texteingabemöglichkeit.
- `cursor: wait`
  Sanduhrsymbol; signalisiert Wartezustand.
- `cursor: help`
  Zeiger mit Fragezeichensymbol, signalisiert Hilfe.
- `cursor: n-resize`, `cursor: s-resize`
  Doppelpfeil nach oben und unten (»north«, »south«)
- `cursor: ne-resize`, `cursor: sw-resize`
  Doppelpfeil nach (»northeast«, »southwest«)
- `cursor: e-resize`, `cursor: w-resize`
  Doppelpfeil nach rechts und links (»east«, »west«)
- `cursor: se-resize`, `cursor: nw-resize`
  Doppelpfeil nach rechts unten und links oben (»southeast«, »northwest«)
- `cursor: url([mauszeigergrafik])`
  die angegebene Grafik dient als Mauszeiger; muss als Datei vom Typ `.cur`, `.ico` oder `.ani` vorliegen.

[zB] *Beispiel:* Das folgende Quelltextbeispiel erzeugt bei Standardeinstellung unter Windows Mauszeiger über den Elementen wie in Abbildung 19.71. Die Mauszeiger sind nachträglich in den Screenshot eingefügt.

```
<div style="float:left;">
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:default">cursor:default</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:crosshair">cursor:crosshair</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:help">cursor:help</div>
```

```
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:move">cursor:move</div>
</div>

<div style="float:left;">
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:pointer">cursor:pointer</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:wait">cursor:wait</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:text">cursor:text</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:n-resize">cursor:n-resize</div>
</div>

<div style="float:left;">
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:e-resize">cursor:e-resize</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:w-resize">cursor:w-resize</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:ne-resize">cursor:ne-resize</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:nw-resize">cursor:nw-resize</div>
</div>

<div style="float:left;">
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:se-resize">cursor:se-resize</div>
<div style="border:solid 1px; width:100px; height:40px; margin:5px;
cursor:sw-resize">cursor:sw-resize</div>
</div>
```

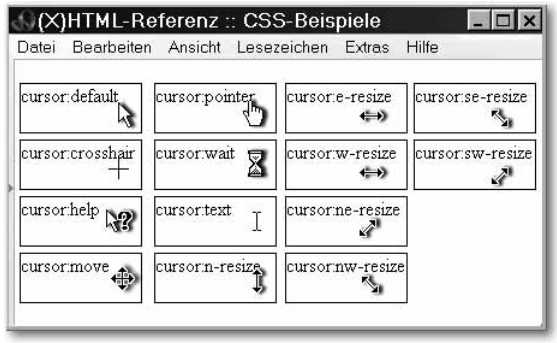

Abbildung 19.71 Verschiedene Cursorstyles bei Mouseover

19.5.11 Contentgenerierung

Mit Hilfe von CSS können nicht nur existierende Inhalte formatiert, sondern auch neu Inhalte neu erzeugt und der Seite hinzugefügt werden. Dies ist in Zusammenhang mit den Pseudo-Elementen `:before` und `:after` sowie bei der Erzeugung von Zählern und Anführungszeichen von Bedeutung.

content

CSS-Level	CSS 2 (geändert in CSS 2.1)
Anwendung	`:before`- und `:after`-Elemente
Vererbung	nein
Werte	*[string], url([grafik]), attr(att-name), [counter], [counters], open-quote, close-quote, no-open-quote, no-close-quote, inherit*
Defaultwert	–

Das `content`-Property wird in Verbindung mit `:before` und `:after` benötigt, um Inhalt vor oder nach dem eigentlichen Elementinhalt weitere Inhalte einzufügen. Dies kann Text sein, jedoch auch (sprachabhängig gewählte) Anführungszeichen, Grafiken oder Zähler.

- `content: "Zeichenkette"`
  fügt die übergebene Zeichenkette ein.
- `content: url("bild.gif")` (in CSS 2.1 abgeschafft)
  fügt die Grafik ein, deren URL übergeben wird.
- `content: counter(countername)`
  fügt den aktuellen Stand eines benannten, mit `counter-increment` definierten Zählers ein.
- `content: counters(countername)`
  fügt den aktuellen Stand der Zähler aller Level mit dem übergebenen Namen ein.
- `content: open-quote`, `content: close-quote`
  fügt ein den Spracheinstellungen entsprechendes, mit dem Property `quotes` definiertes öffnendes oder schließendes Anführungszeichen ein.
- `content: no-open-quote`, `content: no-close-quote`
  reduziert oder erhöht die Hierarchiezählung der Anführungszeichenverschachtelung, ohne dabei jedoch Inhalte einzufügen.

[zB] *Beispiel:*

```
p.anm:before { content: "Anmerkung: " }
```

counter-increment

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[(identifier) (Ganzzahl)?]+, none, inherit*
Defaultwert	*0*

Das `counter-increment`-Property benennt ein oder mehrere Zählervariablen, die in Zusammenhang mit dem Auftreten eines Elements ab Beginn des Quelltextes hochgezählt werden sollen und legt in Form einer Ganzzahl die Schrittweite (Inkrement) des Zählers fest. Wird kein anderslautender Wert für das Inkrement übergeben, so wird der Zähler jeweils um 1 hochgezählt. Es dürfen auch negative Zahlen übergeben werden.

Beispiel: Für dieses Beispiel werden Zähler definiert, die in Zusammenhang mit Überschriften `<h1>` und `<h2>` ausgegeben werden sollen. Für jede Überschrift `<h1>` wird der Zähler `level1` um 1 hochgezählt, für jede Überschrift `<h2>` der Zähler `level2`. **[zB]**

Damit die Zählung auf der zweiten Ebene nach jeder Überschrift `<h1>` erneut beginnt, wird hier der Zähler `level2` stets durch `counter-reset` zurückgesetzt.

```
/* im Stylesheet: */

h1:before { content:counter(level1)'. ';
            counter-increment:level1 1;
            counter-reset:level2;
          }

h2:before { content:counter(level1)'.'counter(level2)'. ';
            counter-increment:level2 1;
          }

<!-- im Dokument: -->
<h1>Kapitel</h1>
<h2>Abschnitt</h2>
<h2>Abschnitt</h2>
<h1>Kapitel</h1>
<h2>Abschnitt</h2>
<h2>Abschnitt</h2>
<h1>Kapitel</h1>
<h2>Abschnitt</h2>
```

Abbildung 19.72 Automatische Kapitelnummerierung in Opera 9.02

counter-reset

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[(identifier) (zielwert)?]+, none, inherit*
Defaultwert	*0*

Das `counter-reset`-Property setzt ein oder mehrere Zählerwerte, deren Namen übergeben werden, auf einen jeweils beliebig festlegbaren ganzzahligen (positiven oder negativen) Wert zurück. Wird kein Wert übergeben, so wird der betreffende Zähler auf 0 gesetzt.

[zB] *Beispiel:*

```
h1:before { content:counter(level1)'. ';
            counter-increment:level1 1;
            counter-reset:level2;
          }

h2:before { content:counter(level1)'.'counter(level2)'. ';
            counter-increment:level2 1;
          }
```

quotes

CSS-Level	CSS 1
Anwendung	global
Vererbung	nein
Werte	*[(string) (string)]+, none, inherit*
Defaultwert	–

Das `quotes`-Property bestimmt die Art der Anführungszeichen, mit dem der Inhalt eines Elementcontainers vom Typ `<q>` oder `<blockquote>` zu umgeben ist. Hierbei werden paarweise die doppelten und einfachen öffnenden und schließenden Anführungszeichen übergeben.

Beispiel: **[zB]**

```
/* im Stylesheet: */

q:lang(de) { quotes: "»" "”"  "," "’"; }

q:lang(fr) { quotes: "«" "»" "‹" "›"; }

<!-- im Dokument: -->
<p><q lang="de">Das hier ist deutsch: <q lang="de">Aenean
fermentum!</q> Metus in lacinia facilisis mauris purus vitae
nibh.</q></p>

<p><q lang="fr">Un petit peu fran&ccedil;ais: <q lang="fr">Aenean
fermentum!</q> Metus in lacinia facilisis mauris purus vitae
nibh.</q></p>
```

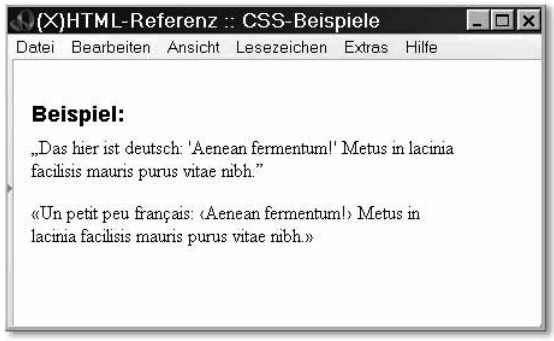

Abbildung 19.73 Sprachabhängiges Setzen von Anführungszeichen (Opera 9.02)

19.5.12 Seitenformatierung (Druckausgabe)

Die CSS-Druckproperties sind in erster Linie für Druckausgabe gedacht, regeln jedoch auch die Ausgabe auf anderen seitenbezogenen Ausgabemedien, beispielsweise Projektionen für Präsentationen.

page

CSS-Level	CSS 2 (nicht mehr in CSS 2.1)
Anwendung	global
Vererbung	nein

Werte	*(identifier), auto*
Defaultwert	*auto*

Mit Hilfe des Properties `page` können einzelne Element für Druckausgabe gesondert formatiert werden. Der gebräuchlichste Fall ist der einer Tabelle mit Überbreite, die im Quer- statt im Längsformat gedruckt werden soll.

[zB] *Beispiel:* Hier wird eine Seite als `"rotated"` definiert, auf die sich die Tabelle bezieht:

```
@page rotated {size: landscape} /* Querformat */)

table.quer {page: rotated; page-break-before: right}
```

page-break-after

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*auto, always, avoid, left, right, inherit*
Defaultwert	*auto*

Das Property `page-break-after` regelt den Seitenumbruch nach bestimmten Elementen. Dieser kann explizit angeordnet oder als »möglichst vermeidbar« gekennzeichnet werden. Ebenso kann der Umbruch bei doppelseitigem Seitenlayout nach einer »linken« oder einer »rechten« Seite erfolgen.

- `page-break-after: always`
  erzwingt einen Seitenumbruch nach dem Element.
- `page-break-after: avoid`
  vermeidet nach Möglichkeit einen Seitenumbruch nach dem Element.
- `page-break-after: left`
  bricht so nach dem Element um, dass das Folgeelement zu Beginn einer »linken« (geradzahligen) Seite erscheint.
- `page-break-after: right`
  bricht so nach dem Element um, dass das Folgeelement zu Beginn einer »rechten« (ungeradzahligen) Seite erscheint.
- `page-break-after: auto` (Default)
  Automatischer Umbruch ohne weitere Berücksichtigung des Elements.

[zB] *Beispiel:*

```
p.last {page-break-after:always}
```

page-break-before

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*auto, always, avoid, left, right, inherit*
Defaultwert	*auto*

Das Property `page-break-before` regelt den Seitenumbruch vor bestimmten Elementen. Dieser kann explizit angeordnet oder als »möglichst vermeidbar« gekennzeichnet werden. Ebenso kann der Umbruch bei doppelseitigem Seitenlayout nach einer »linken« oder einer »rechten« Seite erfolgen.

- `page-break-before: always`
  erzwingt einen Seitenumbruch vor dem Element.
- `page-break-before: avoid`
  vermeidet nach Möglichkeit einen Seitenumbruch vor dem Element.
- `page-break-before: left`
  bricht so vor dem Element um, dass dieses zu Beginn einer »linken« (geradzahligen) Seite erscheint.
- `page-break-before: right`
  bricht so vor dem Element um, dass dieses zu Beginn einer »rechten« (ungeradzahligen) Seite erscheint.
- `page-break-before: auto` (Default)
  Automatischer Umbruch ohne weitere Berücksichtigung des Elements.

Beispiel: **[zB]**

```
h1 {page-break-before:right}
```

page-break-inside

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*avoid, auto, inherit*
Defaultwert	*auto*

Das Property `page-break-inside` regelt den Seitenumbruch innerhalb des Textinhalts eines Elements. Dieser kann durch das Schlüsselwort `"avoid"` als »möglichst zu vermeiden« bestimmt werden.

- `page-break-inside: auto` (Default)
  Automatischer Umbruch, der auch innerhalb des Elementinhalts erfolgen darf
- `page-break-inside: avoid`
  verbietet einen Seitenumbruch innerhalb des Elementinhalts und verschiebt dieses bei Bedarf komplett auf die Folgeseite.

[zB] *Beispiel:*

```
table {page-break-inside:avoid}
```

orphans

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[zahlenwert], inherit*
Defaultwert	*2*

Das Property `orphans` regelt das Auftreten der typographisch als »Schusterjungen« (engl. *orphans*) bezeichneten Anfangszeilen eines Textabsatzes, die durch einen unglücklich erfolgten Seitenumbruch auf der Vorgängerseite verbleiben.

Als numerischer Wert übergeben wird die Mindestanzahl von Zeilen eines Absatzes, die bei einen Seitenumbruch auf der Vorgängerseite verbleiben müssen (Defaultwert: 2). Wird diese Anzahl unterschritten, so soll der gesamte Absatz auf die Folgeseite verschoben werden.

widows

CSS-Level	CSS 2
Anwendung	global
Vererbung	nein
Werte	*[zahlenwert], inherit*
Defaultwert	*2*

Das Property `widows` regelt das Auftreten der typographisch als »Hurenkinder« (engl. *widows*) bezeichneten Schlusszeilen eines Textabsatzes, die durch einen unglücklich erfolgten Seitenumbruch auf der Folgeseite landen.

Als numerischer Wert übergeben wird die Mindestanzahl von Zeilen eines Absatzes, die durch einen Seitenumbruch auf die Folgeseite gelangen müssen (Defaultwert: 2). Wird diese Anzahl unterschritten, so soll der gesamte Absatz auf die Folgeseite verschoben werden.

19.6 Browserspezifische Properties

19.6.1 Scrollbar-Properties (Internet Explorer und Opera)

Die `scrollbar`-Properties werden zwar vom W3C nicht unterstützt (sie werden vermutlich auch nie Teil des Standards werden), erfreuen sich aber seit ihrer Einführung im Internet Explorer gewisser Beliebtheit. Inzwischen beginnen auch andere Browserhersteller, diese proprietären Standards zu unterstützen.

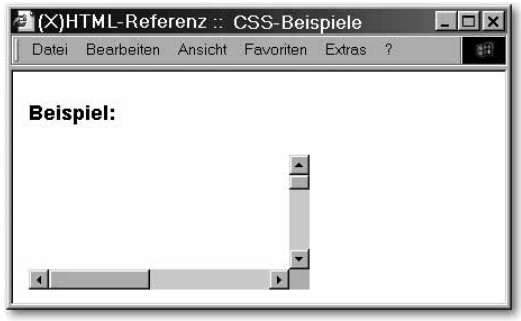

Abbildung 19.74 Ein -Container mit Standardscrollleisten im IE

Die Stylingmöglichkeiten sind von »dezent« bis »knallig« relativ weitreichend.[8] Für den folgenden Screenshot in Opera 8.5 (Abbildung 19.75) sind eine Reihe von Properties gleichzeitig angewendet:

```
div.scroll { scrollbar-face-color:#cccccc;
             scrollbar-3dlight-color:#aaaaaa;
             scrollbar-shadow-color:#aaaaaa;
             scrollbar-track-color:#eeeeee;
             scrollbar-arrow-color:white;
             }
```

Abbildung 19.75 Ein -Container mit Scrollleisten in Opera, leicht gestylt

8 Leider lassen sich die Möglichkeiten nur sehr eingeschränkt im Schwarz-Weiß-Druck zeigen. Für dieses und die nachfolgenden Beispiele gilt deshalb: Am besten probieren Sie die Stylingmöglichkeiten selbst aus.

[»] *Hinweis:* Die Anwendung der `scrollbar`-Properties ist gefahrlos, da sie sich nicht auf das Seitenlayout bezieht. Browser, die diese Properties nicht unterstützen, ignorieren die Anweisungen einfach.

scrollbar-3dlight-color

CSS-Level	*– (Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-3dlight-color` bestimmt die Lichtfarbe der linken und oberen Seite der 3D-Elemente der Scrollbar.

[zB] *Beispiel:*

```
scrollbar-3dlight-color:red;
```

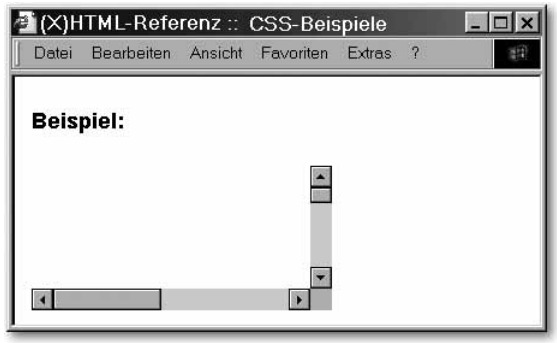

Abbildung 19.76 Highlightfarbe mit scrollbar-3dlight-color

scrollbar-arrow-color

CSS-Level	*– (Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-arrow-color` bestimmt die Farbe des Scrollbar-Pfeils.

Beispiel: **[zB]**

```
scrollbar-arrow-color:white;
```

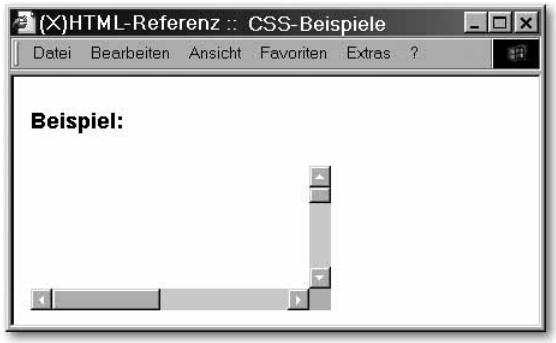

Abbildung 19.77 Farbe des Scrollpfeiles mit scrollbar-arrow-color

scrollbar-base-color

CSS-Level	– *(Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-base-color` bestimmt die Grundfarbe der Scrollbar.

Beispiel: **[zB]**

```
scrollbar-base-color:red;
```

Abbildung 19.78 Grundfarbe der Scrollbar mit scrollbar-base-color

scrollbar-darkshadow-color

CSS-Level	*– (Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-darkshadow-color` bestimmt die dunkle Schattenfarbe der rechten und unteren Seite der 3D-Elemente der Scrollbar.

[zB] *Beispiel:*

```
scrollbar-darkshadow-color:red;
```

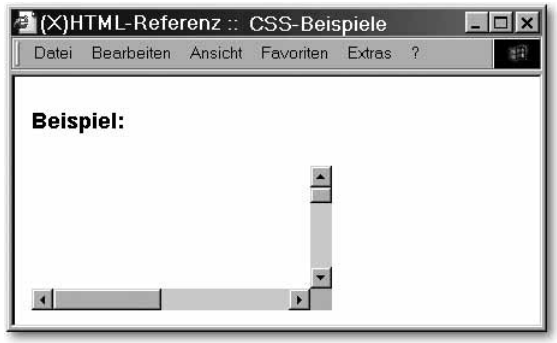

Abbildung 19.79 Dunkle Schattenfarbe mit scrollbar-darkshadow-color

scrollbar-face-color

CSS-Level	*– (Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-face-color` bestimmt die allgemeine Farbe der Scrollbar-Oberfläche.

[zB] *Beispiel:*

```
scrollbar-face-color:red;
```

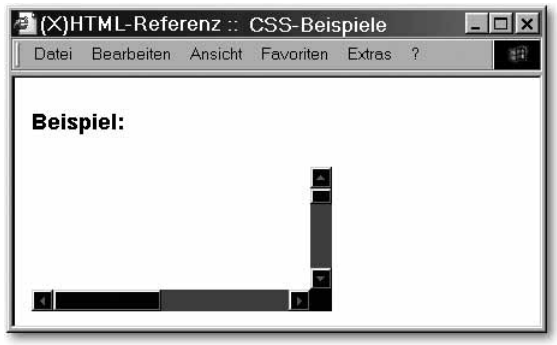

Abbildung 19.80 Oberflächenfarbe der Scrollbar mit scrollbar-face-color

scrollbar-highlight-color

CSS-Level	– *(Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-highlight-color` bestimmt die Lichtfarbe der linken und oberen Seite der 3D-Elemente der Scrollbar.

Beispiel: **[zB]**

```
scrollbar-highlight-color:red;
```

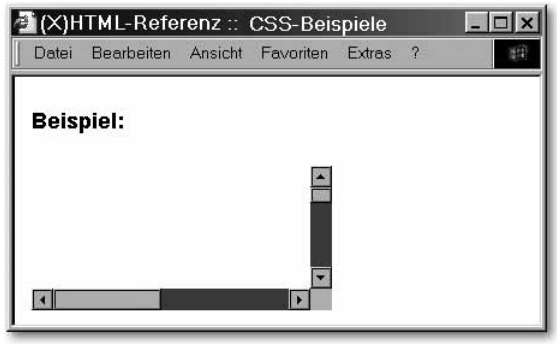

Abbildung 19.81 Lichtkante und Trackfarbe mit scrollbar-highlight-color

scrollbar-shadow-color

CSS-Level	*– (Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-shadow-color` bestimmt die Schattenfarbe an der rechten und unteren Seite der 3D-Elemente der Scrollbar.

[zB] *Beispiel:*

```
scrollbar-shadow-color:red;
```

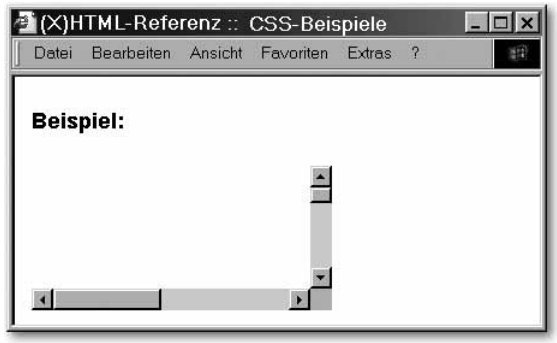

Abbildung 19.82 Schattenfarbe mit scrollbar-shadow-color

scrollbar-track-color

CSS-Level	*– (Microsoft)*
Anwendung	`<html>`, `<bdo>` sowie Elemente mit Scrollbalken: `<applet>`, `<body>`, `<div>`, `<embed>`, `<object>`, `<textarea>`
Vererbung	nein
Werte	*[farbname], [farbwert]*
Defaultwert	–

Das Property `scrollbar-track-color` bestimmt die Hintergrundfarbe der Scrollbarspur (*track*).

[zB] *Beispiel:*

```
scrollbar-track-color:red;
```

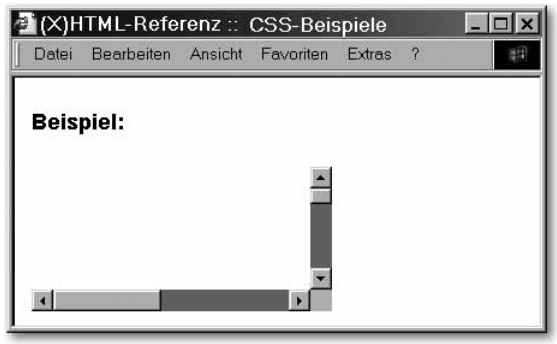

Abbildung 19.83 Trackfarbe mit scrollbar-track-color

19.7 Doctype-Switching und das CSS-Boxmodell

In den Anfängen des Webs konnte man davon ausgehen, dass ein Browser eine Doctype-Deklaration ohnehin nicht beachten würde. Daher war es irrelevant, ob eine derartige Deklaration vorhanden ist oder nicht. Diese Zeiten sind vorbei – moderne Browser beachten durchaus die An- oder Abwesenheit sowie den Inhalt der Doctype-Deklaration. Und sie richten ihr Verhalten entsprechend aus: Das hierbei verfolgte Prinzip nennt sich **Doctype-Switching**.

Was ist Doctype-Switching?

Mit dem Begriff Doctype-Switching beschreibt man das Umschalten des Browserverhaltens beim Darstellen einer Webseite anhand der vorgefundenen Doctype-Deklaration. Ein und dieselbe HTML-Seite wird demnach unterschiedlich dargestellt, je nachdem, welche Art von Deklaration sie besitzt.

Welche Browser betreiben Doctype-Switching?

Zu den Browsern, die Doctype-Switching vornehmen, gehören

- Internet Explorer 6 und höher, Opera 7 und höher (Windows)
- Mozilla, Netscape 6 und höher, Firefox (Windows, Linux)
- Galeon, Konqueror (Linux)
- Internet Explorer 5, Camino, Safari (MacOS)

Grundlage sind verschiedene, im Browser implementierte **Layout-Modi**, die zwischen solchen Seiten unterscheiden, die auf Darstellungseigenheiten (»Quirks«) früherer Browser hin entworfen sind, also nicht standardkonformes HTML oder

»*transitional*«-Vokabular einsetzen, und solchen, die gemäß dem Dokumenttyp »*strict*« aufgebaut sind und sich nach üblichen Standards richten.

Verschiedene Layout-Modi

Man unterscheidet in der Theorie zwei offizielle, in der Praxis für manche Browser sogar insgesamt drei Layout-Modi:

- **Quirks-Mode** (»BackCompat Mode«)
  In diesem Modus rendert der Browser nicht streng standardkonform, sondern emuliert das Darstellungsverhalten älterer Browser. So wird es ermöglicht, auch ältere Seiten darzustellen, die um Browserbugs herum designt wurden (»Tag Soup«-Seiten).

  Das bedeutet: Mozilla bildete anfänglich das Verhalten von Netscape 4.x nach, der Internet Explorer 6.0 dasjenige von Internet Explorer 5.0.

- **Standards-Mode** (»Standards compliance mode«, «CSS1Compat Mode«)
  In diesem Modus wird eine Seite streng gemäß der vorgeschriebenen Standards dargestellt – denen sie dann aber auch entsprechen sollte …

Der Quirks- und der Standards-Modus betreffen im Wesentlichen das **CSS-Boxmodell**, dessen Umsetzung in älteren Browsern, gelinde gesagt, unzureichend war. Die den Webdesignern aufgezwungenen Umwege führen jedoch dazu, dass entsprechend entworfene Seiten in standardkonformen Browsern fehlerhaft dargestellt werden. Hier ist die Möglichkeit der gezielten Wahl eines Darstellungsmodus sinnvoll.

Neben dem CSS-Boxmodell existiert jedoch noch eine weitere Fehlerquelle, die im nicht standardisierten Einsatz von **Tabellenabmessungen** besteht – speziell geht es hier um die vertikalen Abmessungen von Tabellen bzw. Tabellenzellen: Das hier beliebte und gern eingesetzte `height`-Attribut ist bei genauer Betrachtung tatsächlich *nicht* Teil der HTML-Spezifikation, wurde bislang aber von den Browsern unterstützt. Da standardkonform arbeitende Browser dem jedoch nicht folgen können, wurde ein eigener, inoffizieller Darstellungsmodus eingerichtet, der diesem Problem abhelfen soll:

- **Beinahe-Standards-Modus** (»Almost Standards Mode«)
  Die CSS-Box wird gemäß der Standards umgesetzt. Die (nicht standardkonforme) vertikale Dimensionierung von Tabellenzellen wird beachtet.

  Gilt für: Mozilla, Safari und Opera ab 7.5. Diese setzen vertikale Dimensionsierung gemäß CSS 2 in ihrem »Standards«-Mode konsequent um.

Die Einigkeit ist nur scheinbar – Vorsicht ist geboten!

Auch wenn bei den Browserherstellern immerhin Einigkeit über die *Unterscheidung* zwischen Darstellungsmodi herrscht, so gilt dies dennoch nicht für deren *Umsetzung* – im Verhalten entsprechen »Quirks«- und »Standards«-Modus von (beispielsweise) Mozilla nicht hundertprozentig den gleichnamigen Modi von (beispielsweise) Internet Explorer oder anderen Browsern. Die Notwendigkeit des Testens entfällt damit in Zukunft also nicht, wenn auch die Unterschiede beim »Standards"-Mode gering ausfallen.

Auswirkungen auf das CSS-Boxmodell

Im Internet Explorer sind die Unterschiede bei Umsetzung des CSS-Boxmodells erheblich (siehe Abbildung 19.84). Im Quirks Mode wird die Angabe der *Innenbreite als Gesamtbreite* interpretiert, von der also Padding- und Borderbreiten *abgezogen* werden. Im Standards Mode zeigt der Internet Explorer das korrekte Verhalten, das in der *Addition dieser Werte* besteht. Die Box ist im linken Fall 300 Pixel breit (falsch) und im rechten 350 Pixel (korrekt).

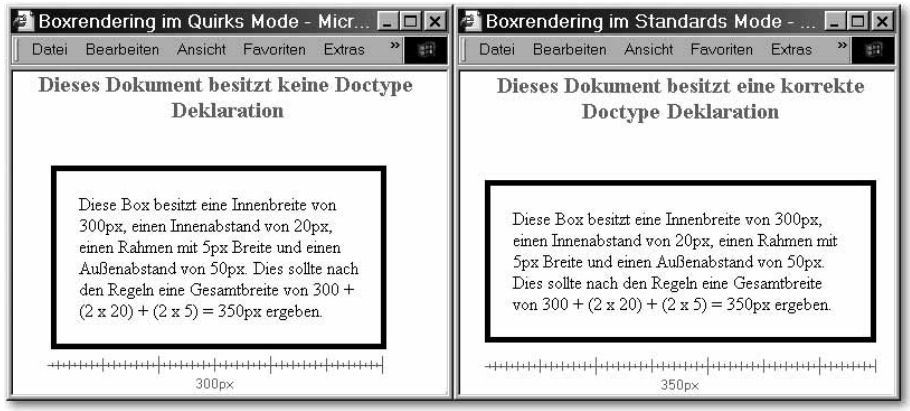

Abbildung 19.84 Internet Explorer: links Quirks Mode, rechts Standards Mode

Folgendes HTML liegt beiden Anzeigen zugrunde. Der einzige Unterschied besteht darin, dass im einen Fall die Doctype-Deklaration entfernt wurde, um den Quirks Mode zu triggern:

```
<!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>Boxrendering im Standards Mode</title>
    <style type="text/css">
```

```
      /* <[![CDATA[ */

      h1  { font-size:20px;
            color:red;
            text-align:center;
      }

      div#box { width:300px;
                padding:20px;
                border:5px solid black;
                margin:50px;
      }

      /* ]]> */
    </style>
  </head>
  <body>
    <h1>Dieses Dokument besitzt eine
     korrekte Doctype-Deklaration</h1>

    <div id="box">Diese Box besitzt eine Innenbreite
     von 300px, einen Innenabstand von 20px, einen Rahmen
     mit 5px Breite und einen Außenabstand von 50px.
     Dies sollte nach den Regeln eine Gesamtbreite von
     300 + (2 x 20) + (2 x 5) = 350px ergeben.
    </div>

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

Anmerkung: Beachten Sie die *Auskommentierung der CDATA-Bereichsbegrenzer* nach CSS-Kommentarsyntax. Dies verhindert Darstellungsprobleme mancher Browser, die CSS-Anweisungen in CDATA-Bereichen gar nicht oder unvollständig interpretieren.

Umschalten des Verhaltens anhand der Doctype-Deklaration

Ein Browser, der verschiedene Darstellungsmodi »bereithält«, wird zunächst die Doctype-Deklaration des darzustellenden Dokuments einlesen und anschließend »entscheiden«, welcher Layout-Modus zu wählen ist.

Beachten Sie, dass der Internet Explorer nur »Standards Mode« und »Quirks Mode« kennt, wohingegen Mozilla, Safari und Opera auf »transitional«-Doctypes

mit »Almost Standards Mode« reagieren. Ohne oder mit zu altem Doctype gilt generell Quirks Mode.

Doctype-Deklaration	Modus
– (keine vorhanden)	Quirks Mode
Doctype-Deklaration (prä HTML 4)	Quirks Mode
vollständige Doctype-Deklaration (HTML 4.01 oder XHTML)	Standards Mode
`PUBLIC`-Identifier *ohne* URI-Angabe der DTD	Quirks Mode
`PUBLIC`-Identifier *mit* URI-Angabe der DTD	Standards Mode
`PUBLIC`-Identifier *mit* beliebiger URI-Angabe	Standards Mode
»transitional«-Doctype (Mozilla, Safari, Opera)	Almost Standards Mode

Tabelle 19.5 Art der Doctype-Deklaration und getriggerter Darstellungsmodus

Neben der Anpassung der Darstellung der Seite verändert sich auch das Verhalten der Browser in Bezug auf *Syntaxfehler in den CSS-Angaben* selbst: Wird hier im »Quirks Mode« noch allerhand verziehen, so muss das CSS im Standards Mode fehlerfrei sein. Fehlerhafte Anweisungen werden ignoriert.

Internet Explorer 6 für Windows will die Doctype-Deklaration am Anfang

Platziert man *vor der Doctype Deklaration* eine irgendwie geartete Zeichenkette, so schaltet der Internet Explorer 6 unabhängig von der verwendeten Doctype Deklaration in den *Quirks-Modus*.

Der Fehler tritt auf, wenn man der Doctype-Deklaration für ein XHTML-Dokument eine *XML-Deklaration* voransetzt (was gestattet sein sollte), aber auch wenn es sich bei den vor der Doctype-Deklaration stehenden Zeichen um einen Kommentar oder gar Leerraum (Zeilenumbrüche, Leerzeichen) handelt.

Im *Internet Explorer 7* existiert dieses Verhalten nicht mehr.

Hinweis: In XML-Dokumenten ist die XML-Deklaration optionaler Teil des Prologs, darf also entfallen. Hier besteht jedoch die analoge (und sinnvolle) Regelung, dass *vor der XML-Deklaration*, falls sie verwendet wird, keine weiteren Zeichen erlaubt sind. Dies ist damit begründet, dass der Parser aus der Zeichenfolge `<?xml` am Dokumentbeginn dessen Zeichencodierung entnehmen soll. **[«]**

Feststellung des Modus einer Seite

Wenn Sie (in Mozilla/Firefox) feststellen wollen, in welchem Modus ein Dokument angezeigt wird, so können Sie dies mit Hilfe der Web Developer Toolbar von Chris Pederick herausfinden (diese ist in Kapitel 10, *Werkzeuge*, genauer beschrieben). Sie können dies über folgenden Menüpunkt anwählen:

Informations > View Page Information > General > Render Mode

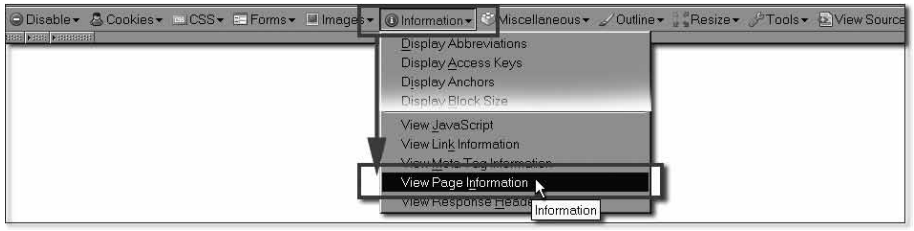

Abbildung 19.85 Render Mode über Web Developer Toolbar (Menü gekürzt)

- **Web Developer Toolbar: http://chrispederick.com/work/webdeveloper/**

19.7.1 Debugging von CSS

Um die Deklarationen eines CSS-Stylesheets auf Korrektheit zu überprüfen, stellt das W3C online einen **CSS-Validator** mit mehrsprachiger Oberfläche (darunter auch Deutsch) zur Verfügung. Sie können eine CSS-Datei online durch Übergabe des URL der mit ihr verknüpften Webseite überprüfen lassen, aber per Upload auch lokal vorliegende CSS-Dateien. Als dritte Option kann CSS-Code direkt in ein Textfeld eingegeben und validiert werden.

- **CSS-Validator: http://jigsaw.w3.org/css-validator/validator.html.de**

Abbildung 19.86 CSS-Validator des W3C (http://jigsaw.w3.org/css-validator/)

Die **Optionswahl** (siehe Abbildung 19.86, unter dem Eingabefeld sichtbar) kann den Validierungsvorgang auf bestimmte CSS-Versionen einschränken (CSS 1, 2, 2.1 oder 3), stellt aber auch verschiedene SVG-Profile, mobile- und TV-´Profile zur Verfügung. Auch CSS für bestimmte Ausgabemedien kann herausgegriffen werden, indem das gewünschte `media`-Attribut ausgewählt wird. Last, but not least kann Art und Umfang der ausgegebenen Fehlermeldungen festgelegt werden. Die Nutzung des Validators ist kostenlos.

Anhang

A **W3C-Ressourcen** 1137

B **Weitere Ressourcen** 1165

C **Inhalt der Begleit-CD** 1177

Dieser Abschnitt dient als Wegweiser durch die Vielzahl von W3C-Spezifikationen, die HTML, XHTML, CSS und verwandte Themen betreffen, jeweils mit kurzer Inhaltsangabe und Einstufung der Richtlinie.

A W3C-Ressourcen

Das World Wide Web Committee, kurz W3C genannt, stellt heute die maßgebliche Instanz dar, wenn es um die Festlegung von Regelwerken und Spezifikationen von Websprachen wie HTML, XHTML und CSS geht.

A.1 W3C-Spezifikationen

Die Spezifikationen des W3C zielen in ihren Endversionen auf »Quasi-Standards« ab, werden aber auch in vorläufigen Arbeitsversionen, den »Working Drafts«, publiziert. Je nach dem Stand der Fertigstellung bzw. des Status der Veröffentlichung werden verschiedene Bezeichnungen verwendet.

Abbildung A.1 Die Quelle der Spezifikationen – www.w3.org

W3C Recommendation

Die Endversion einer Empfehlung, die einem anerkannten Standard gleichgesetzt werden kann. Es ist davon auszugehen, dass die Spezifikation weitgehend unterstützt und implementiert wird. Eine Recommendation kann modifiziert oder (selten) verworfen werden, gilt ansonsten aber als stabil und normativ.

W3C Proposed Recommendation

Die unmittelbare Vorstufe zu einer offiziellen Empfehlung. Kann als reifer Standard betrachtet werden, der bereits ausreichend geprüft wurde, eingereicht zur endgültigen Verabschiedung durch das W3C Advisory Committee. Änderungen sind zu diesem Zeitpunkt unwahrscheinlich.

W3C Proposed Edited Recommendation

Eine vom W3C bearbeitete Empfehlung, die von der Community auf Stichhaltigkeit und Konformität geprüft werden soll, bevor sie als Recommendation veröffentlicht wird.

W3C Candidate Recommendation

Eine »Beinahe-Empfehlung«, die in der vorliegenden Form die mit dem Thema befasste Arbeitsgruppe bereits zufrieden stellt und von der abschließende Erfahrungen mit Implementierungen gesammelt werden sollen. Anschließend sind Änderungen auf dieser Grundlage noch denkbar.

W3C Working Draft

Eine Arbeitsversion der Spezifikation, die der Öffentlichkeit zwecks Kommentar und als Diskussionsgrundlage zur Verfügung gestellt wird. Sie unterliegt in der Regel bis zur endgültigen Spezifikation noch erheblichen Änderungen und sollte daher nicht als Grundlage für Implementierungen verwendet werden.

W3C Working Group Note

Veröffentlichung einer Arbeitsgruppe, sobald das bearbeitete Thema als abgeschlossen betrachtet wird; gilt als nicht normativ. Kann, muss jedoch nicht, mit der Veröffentlichung eines Working Draft einhergehen.

Alle W3C Veröffentlichungen

Eine vollständige Liste aller W3C Veröffentlichungen (Recommendations, Working Drafts, Notes und zurückgezogene Arbeiten) befindet sich unter **www.w3.org/TR/**.

A.1.1 W3C Ressourcen zu HTML

ISO HTML (non-W3C)

Status
ISO Standard (ISO/IEC), 15. Mai 2000 (Revision 2003)
Beschreibung
Der als »ISO HTML« bezeichnete Standard ISO/IEC 15445:2000 stellt eine von der Sprachgrammatik her rigider formulierte Version des ansonsten als normativ anerkannten HTML 4.01 dar.
Herausgeber
ISO/IEC (Roger Price, David M. Abrahamson)
Website
www.cs.tcd.ie/15445/15445.HTML

HTML 4.01

Status
W3C Recommendation, 24. Dezember 1999
Beschreibung
Aktueller Standard. HTML in der Sprachversion 4.01 mit gegenüber den Vorgängerversionen HTML 3.2 und 2.0 erweiterten Multimediafähigkeiten; geringfügige Fehlerbereinigungen gegenüber HTML 4.0; liegt in drei Versionen vor (strict, transitional, frameset); **Ablösung von HTML 4.0**
Herausgeber
Dave Raggett, Arnaud Le Hors, Ian Jacobs
Website
www.w3.org/TR/html401/

HTML 4.0

Status
Historisch; W3C Recommendation, Revision 24. April 1998
Beschreibung
HTML in der Sprachversion 4.0 mit gegenüber den Vorgängerversionen HTML 3.2 und 2.0 erweiterten Multimediafähigkeiten; Ablösung von HTML 3.2; erstmals mehrere Versionen (strict, transitional, frameset).
Herausgeber
Dave Raggett, Arnaud Le Hors, Ian Jacobs
Website
www.w3.org/TR/1998/REC-html40–19980424/

HTML 3.2

Status
Historisch; W3C Recommendation, 14. Januar 1997
Beschreibung
HTML in der Sprachversion 3.2 mit Erweiterungen gegenüber den Vorgängerversionen HTML 2.0; Ablösung von HTML 2.0; Kompromiss zwischen HTML 3.0 und den durch die Browserindustrie geschaffenen Fakten.
Herausgeber
Dave Raggett
Website
www.w3.org/TR/REC-html32

HTML 3.0

Status
Historisch; Gelöschte W3C Recommendation, März 1995
Beschreibung
W3C-Vorschlag als Nachfolgeversion zu HTML 2.0. Basierte auf HTML+; zurückgezogen zugunsten von HTML 3.2 (Löschung der Recommendation am 28. September 1995).
Herausgeber
Dave Raggett
Website
www.w3.org/MarkUp/html3/

HTML 2.0

Status
Historisch; W3C Recommendation, 21. September 1995 (Revision)
Beschreibung
HTML in der Sprachversion 2.0; erstmals auf einer SGML-DTD basierend; veröffentlicht als RFC1866
Herausgeber
Tim Berners-Lee, Daniel Connolly
Website
www.w3.org/MarkUp/html-spec/html-pubtext.html alternativ (als RFC 1866): ftp.ics.uci.edu/pub/ietf/html/rfc1866.txt

HTML+ (HTMLPlus)

Status
Historisch; Studie, 8. November 1993
Beschreibung
Diskussionsgrundlage für eine denkbare Weiterentwicklung von HTML (1.0); führte Tabellen und Formulare ein und diente später teilweise als Grundlage des ebenfalls nicht verwirklichten HTML 3.0.
Herausgeber
Dave Raggett
Website:
www.w3.org/MarkUp/HTMLPlus/htmlplus_1.html

HTML 1.0

Status
Historisch; Internet Draft, IIIR Working Group, Juni 1993
Beschreibung
Öffentliche Darstellung von HTML – noch ohne Nennung einer Versionsnummer, keine eigentliche Spezifikation
Herausgeber
Tim Berners-Lee, Daniel Connolly
Website
www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt

A.1.2 W3C Ressourcen zu XHTML 1

XHTML 1.1– Module-based XHTML (Second Edition)

Status
W3C Working Draft, 16. Februar 2007
Beschreibung
Neufassung des aktuellen Standards. Im Wesentlichen (mit geringfügigen Änderungen) eine Neuformulierung von XHTML 1.0 strict unter Verwendung von XHTML Modularization 1.0; entsprechend durch Hinzunahme weiterer Module erweiterbar;
Herausgeber
Murray Altheim, Shane McCarron, Masayasu Ishikawa
Website
www.w3.org/TR/xhtml11/

XHTML 1.0

Status
W3C Recommendation 26. Januar 2000, Revision 1. August 2002
Beschreibung
Aktueller Standard. Eine Neuformulierung der drei DTD-Versionen (strict, transitional, frameset) von HTML 4.01 als XML-Anwendungen; die aktuelle Revision berücksichtigt seit Januar 2000 gefundene Errata.
Herausgeber
Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler (2nd Ed.), François Yergeau (3rd Ed.)
Website
www.w3.org/TR/xhtml1/

XHTML Modularization 1.1

Status
W3C Working Draft, 5. Juli 2006
Beschreibung
Definition der Aufteilung (*decomposition*) von XHTML 1.0 (!) in abstrakte Module, als Grundlage verschiedener Markupsprachen; die Module sind mit DTDs und auch XML-Schema definiert.
Herausgeber
Daniel Austin, Subramanian Peruvemba, Shane McCarron, Masayasu Ishikawa
Website
www.w3.org/TR/xhtml-modularization/

XHTML Modularization 1.0

Status
W3C Recommendation, 10. April 2001
Beschreibung
Aktueller Standard. Definition der Aufteilung (*decomposition*) von XHTML 1.0 in abstrakte Module (nur DTDs), als Grundlage verschiedener Markupsprachen; wird demnächst durch XHTML Modularization 1.1 abgelöst
Herausgeber
Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer, Ted Wugofski
Website
www.w3.org/TR/2001/REC-xhtml-modularization-20010410/

Modularization of XHTML in XML Schema

Status
W3C Working Draft, 3. Oktober 2003
Beschreibung
Beschreibt die Modularisierung von XHTML mit Hilfe von XML-Schema und auf XML-Schema basierende Implementierungen der Module aus »XHTML Modularization«. Last Call endete 14. November 2003. Aufgegangen in XHTML Modularization 1.1.
Herausgeber
Daniel Austin, Subramanian Peruvemba, Shane McCarron, Masayasu Ishikawa
Website
www.w3.org/TR/xhtml-m12n-schema

XHTML 1.0 in XML Schema

Status
W3C Note, 2. September 2002
Beschreibung
Beschreibt XHTML 1.0 mit Hilfe von XML-Schema (zu jener Zeit noch im Frühstadium). Eingestuft als »informativ«.
Herausgeber
Masayasu Ishikawa
Website
www.w3.org/TR/xhtml1-schema

XHTML Basic 1.1

Status
W3C Working Draft 5. Juli 2006
Beschreibung
Revision von XHTML Basic 1.0; basiert auf der Modularisierung von XHTML; beschreibt die minimale Modulkombination für eine XHTML Host Language.
Herausgeber
Shane McCarron, Masayasu Ishikawa, Mark Baker, Shinichi Matsui, Peter Stark, Ted Wugofski, Toshihiko Yamakami
Website
www.w3.org/TR/xhtml-basic/

XHTML Basic 1.0

Status
W3C Recommendation, 19. Dezember 2000
Beschreibung
Basiert auf der Modularisierung von XHTML; beschreibt die minimale Modulkombination für eine XHTML Host Language, wie sie auch als Grundlage für Mobilanwendungen (Mobiltelefone, Pager, PDAs) dienen kann.
Herausgeber
Mark Baker, Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Ted Wugofski, Toshihiko Yamakami
Website
www.w3.org/TR/xhtml-basic/

XHTML-Print

Status
W3C Recommendation, 20. September 2006
Beschreibung
Variante von XHTML Basic; basiert auf der Modularisierung von XHTML; ermöglicht Drucken auf einfachen Druckern von beispielsweise Mobilgeräten aus bzw. in Umgebungen, in denen keine speziellen Druckertreiber zur Verfügung stehen; steht in Verbindung mit der Spezifikation »CSS Print Profile«.
Herausgeber
Melinda Grant, Jim Bigelow
Website
www.w3.org/TR/xhtml-print/

XHTML Media Types

Status
W3C Note, 1. August 2002
Beschreibung
Diskutiert den Einsatz von Media Types (Mimetypes) für XHTML-Dokumente; Gegenüberstellung der Verwendung von `application/xhtml+xml`, `application/xml`, `text/xml` und `text/html`.
Herausgeber
Ishikawa Masayasu
Website
www.w3.org/TR/xhtml-media-types/xhtml-media-types.html

HLink – Link recognition for the XHTML Family

Status
W3C Working Draft, 13. September 2002
Beschreibung
Ermöglicht für Sprachen der XHTML-Famile in Modulform das Hinzufügen von Hyperlink-attributen inklusive der Beschreibung der Linktraversierung; erweitert die Möglichkeiten von XLink.
Herausgeber
Steven Pemberton, Masayasu Ishikawa
Website
www.w3.org/TR/hlink

XHTML + MathML + SVG Profile

Status
W3C Working Draft, 9. August 2002
Beschreibung
Die Verbindung aus XHTML, MathML und SVG erlaubt, durch Einsatz des XML-Namens-raummechanismus die Verwendung aller drei Markupsprachen in einem validierbaren Dokument.
Herausgeber
W3C HTML Working Group, SVG Working Group
Website
www.w3.org/TR/XHTMLplusMathMLplusSVG/xhtml-math-svg.html

A.1.3 W3C Ressourcen zu XHTML 2

XHTML 2.0

Status
W3C Working Draft, 26. Juli 2006
Beschreibung
Konzipiert als Allzwecksprache für das World Wide Web, basierend auf XHTML 1.0/1.1, aber mit umfassenden Neuerungen der Strukturdarstellung und Bereinigungen; ebenfalls in Form abstrakter Module definiert.
Herausgeber
Jonny Axelsson, Mark Birbeck, Micah Dubinko, Beth Epperson, Masayasu Ishikawa, Shane McCarron, Ann Navarro, Steven Pemberton
Website
www.w3.org/TR/xhtml2/

A.1.4 W3C Ressourcen zu CSS 1 und 2

CSS Level 2.1 (Level 2 Revision 1)

Status
W3C Working Draft 11. April 2006
Beschreibung
Überarbeitung von CSS 2; nimmt als Fehlerbereinigung einige Teile der Spezifikation zurück; soll im Endeffekt CSS 2.0 ablösen.
Herausgeber
Bert Bos, Tantek Çelik, Ian Hickson, Håkon Wium Lie
Website
www.w3.org/TR/CSS21

CSS Level 2

Status
W3C Recommendation, 12. Mai 1998
Beschreibung
Aktueller Standard. Definition von Cascading Style Sheets Level 2; baut auf Level 1 auf; wird in allen aktuellen Browsern weitestgehend unterstützt.
Herausgeber
Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs
Website
www.w3.org/TR/REC-CSS2

CSS Level 1

Status
W3C Recommendation, 17. Dezember 1996, Revision 11. Januar 1999
Beschreibung
Aktueller Standard. Definition von Cascading Style Sheets Level 1; wird in allen aktuellen Browsern unterstützt
Herausgeber
Håkon Wium Lie, Bert Bos
Website
www.w3.org/TR/REC-CSS1

A.1.5 W3C Ressourcen zu CSS 3

CSS Level 3 (Introduction to CSS3)

Status
W3C Working Draft, 23. Mai 2001
Beschreibung
CSS Level 3 ist in zahlreiche Einzelmodule untergliedert, von denen einige inhaltlich auf CSS Level 2 basieren, vom Umfang teilweise jedoch erheblich erweitert wurden. Dieses Dokument bietet als Roadmap einen Überblick, ist allerdings weniger aktuell als die Beschreibungen der Einzelmodule.
Herausgeber
Eric A. Meyer, Bert Bos
Website
www.w3.org/TR/css3-roadmap/

CSS3 Advanced Layout Module

Status
W3C Working Draft, 15. Dezember 2005
Beschreibung
Behandelt fortgeschrittene Aspekte der Positionierung und Ausrichtung in GUIs oder Screen-Layouts unter Repositionierung von Elementen gegenüber der Quelltextreihenfolge.
Herausgeber
Bert Bos
Website
www.w3.org/TR/css3-layout/

CSS3 Backgrounds and Borders Module

Status
W3C Working Draft, 16. Februar 2005
Beschreibung
Behandelt die CSS3-Funktionalitäten zur Beschreibung von Hintergründen und Rändern von Elementen wie multiple Hintergrundbilder oder den Einsatz von Grafiken als Border; beinhaltet und erweitert die entsprechenden Funktionalitäten aus CSS 1 und 2. (Ersetzt die älteren Drafts »CSS3 Borders« und »CSS3 Backgrounds«)
Herausgeber
Tim Boland, Bert Bos
Website
www.w3.org/TR/css3-background/

CSS3 Box Model Module

Status
W3C Working Draft, 24. Oktober 2002
Beschreibung
Erläutert, aufbauend auf dem CSS3 Text Modul, das entsprechende CSS3-Boxmodell; abwärtskompatibel zu CSS 2, erweitert dieses jedoch.
Herausgeber
Bert Bos
Website
www.w3.org/TR/css3-box/

CSS3 Cascading and Inheritance Module

Status
W3C Working Draft, 15. Dezember 2005
Beschreibung
Beschreibt das Cascading-Prinzip bei der Wertzuweisung an Properties von Elementen; entspricht mit geringfügigen Änderungen der Funktionalität von CSS2.
Herausgeber
Håkon Wium Lie
Website
www.w3.org/TR/css3-cascade/

CSS3 Color Module

Status
W3C Candidate Recommendation, 14. Mai 2003
Beschreibung
Properties und Werte in Zusammenhang mit Farbgebung; basiert erweiternd auf CSS2.
Herausgeber
Tantek Çelik, Chris Lilley
Website
www.w3.org/TR/css3-color/

CSS3 Fonts Module

Status
W3C Working Draft, 2. August 2002
Beschreibung
Properties und Werte in Zusammenhang mit Schrift, wie Schriftarten, Hervorhebungen, Textdekoration; basiert erweiternd auf CSS2.

Herausgeber
Michel Suignard, Chris Lilley
Website
www.w3.org/TR/css3-fonts/

CSS3 Generated and Replaced Content Module

Status
W3C Working Draft, 14. Mai 2003
Beschreibung
Generierung oder Verschiebung von Inhalten zur Darstellung von Fußnoten, Anmerkungen, Kopf- und Fußbereichen; Behandlung der Erzeugung automatischer Nummerierungen und von Techniken zum Ersetzen oder Beschneiden von Grafiken.
Herausgeber
Ian Hickson
Website
www.w3.org/TR/css3-content/

CSS3 Generated Content for Paged Media

Status
W3C Working Draft, 5. Februar 2007
Beschreibung
Beschreibt die Generierung von Fußnoten, Anmerkungen, Kopf- und Fußbereichen; Behandlung der Erzeugung automatischer Nummerierungen (ersetzt möglicherweise das Generated and Replaced Content Module).
Herausgeber
Håkon Wium Lie
Website
www.w3.org/TR/css3-gcpm

CSS3 Hyperlink Presentation Module

Status
W3C Working Draft, 24. Februar 2004
Beschreibung
Festlegung der Rolle von Hyperlinks; speziell in Bezug auf Festlegung des Zielfensters (`target`-Property).
Herausgeber
Tantek Çelik, Bert Bos, Daniel Glazman

Website
www.w3.org/TR/css3-hyperlinks/

CSS3 Line Module

Status
W3C Working Draft, 15. Mai 2002
Beschreibung
Schlägt eine Reihe von Zeilenformatierungsmöglichkeiten vor (»Inline Element Box Model«); behandelt Grundlinienausrichtung, und Pseudoelemente wie `::first-line` und `::first-letter`; beinhaltet und erweitert die entsprechenden Funktionalitäten von CSS2.
Herausgeber
Michel Suignard, Eric A. Meyer
Website
www.w3.org/TR/css3-linebox/

CSS3 Lists Module

Status
W3C Working Draft, 7. November 2002
Beschreibung
Behandelt die Anwendung von CSS-Styles auf Listen; beinhaltet und erweitert die entsprechenden Funktionalitäten von CSS2.
Herausgeber
Ian Hickson, Tantek Çelik
Website
www.w3.org/TR/css3-lists/

CSS3 Media Queries Module

Status
W3C Candidate Recommendation, 8. Juli 2002
Beschreibung
Behandelt Media Types und den medienabhängigen Einsatz von Stylesheets; Erweiterung betreffender Möglichkeiten aus CSS2.
Herausgeber
Håkon Wium Lie, Tantek Çelik, Daniel Glazman
Website
www.w3.org/TR/css3-mediaqueries/

CSS3 Multi-column Layout Module

Status
W3C Working Draft, 15. Dezember 2005
Beschreibung
Beschreibt mehrspaltiges Layout in CSS; erweitert hierfür das Boxmodell, so dass Inhalte eines Elements in mehrere Spalten fließen können.
Herausgeber
Håkon Wium Lie
Website
www.w3.org/TR/css3-multicol/

CSS3 Paged Media Module

Status
W3C Working Draft, 10. Oktober 2006
Beschreibung
Basiert auf dem CSS3 Box Model Modul; beschreibt die Aufteilung eines Flowinhalts auf Druckseiten; führt Header und Footer sowie druckspezifische Erweiterungen für generierten Content ein.
Herausgeber
Håkon Wium Lie, Jim Bigelow
Website
www.w3.org/TR/css3-page/

CSS3 Presentation Levels Module

Status
W3C Working Draft, 13. August 2003
Beschreibung
Behandelt so genannte Präsentations-Level, die das Verhalten von Dokumenten für verschiedene Präsentationen steuern können (Gliederungsansichten, Präsentation mit Seitenübergängen oder Animationen u.ä.)
Herausgeber
Ian Hickson, Håkon Wium Lie
Website
www.w3.org/TR/css3-preslev/

CSS3 Ruby Module

Status
W3C Candidate Recommendation, 14. Mai 2003
Beschreibung
Definiert die Darstellung von Ruby-Elementen mittels CSS und das Ruby-Boxmodell.
Herausgeber
Marcin Sawicki, Michel Suignard
Website
www.w3.org/TR/css3-ruby/

CSS3 Selectors Module

Status
W3C Working Draft, 15. Dezember 2005
Beschreibung
Erhebliche Erweiterung der Selektoren aus CSS1 und 2, Optimierung für XML und HTML; derzeit bereits in Ansätzen unterstützt.
Herausgeber
Daniel Glazman, Tantek Çelik, Ian Hickson, Peter Linss, John Williams
Website
www.w3.org/TR/css3-selectors/

CSS3 Speech Module

Status
W3C Working Draft, 16. Dezember 2004
Beschreibung
Properties und Werte für Sprachausgabe mit CSS3; berücksichtigt die Spezifikation der Speech Synthesis Markup Language (SSML, siehe dort).
Herausgeber
Dave Raggett, Daniel Glazman, Claudio Santambrogio
Website
www.w3.org/TR/css3-speech/

CSS3 Syntax Module

Status
W3C Working Draft, 13. August 2003
Beschreibung
Behandelt die Grundlagen der Struktur, der Syntax und der Regeln des Parsings von CSS Stylesheets; darüber hinaus (informativ) die medienabhängige Verknüpfung von Styles und Dokumenten.
Herausgeber
L. David Baron
Website
www.w3.org/TR/css3-syntax/

CSS3 Text Effects Module

Status
W3C Working Draft, 27. Juni 2005
Beschreibung
Properties und Werte zur Textdarstellung; behandelt Weißraum, Zeilenumbrüche, Textausrichtung und Spationierung.
Herausgeber
Elika J. Etemad, Michel Suignard
Website
www.w3.org/TR/css3-text/

CSS3 User Interface Module

Status
W3C Working Draft, 16. Februar 2000 (Entwicklung eingestellt)
Beschreibung
Pseudoklassen, Properties und Werte mit Bezug zur Darstellung des User Interfaces; erweitert entsprechende Elemente aus CSS 2. **Abgelöst durch CSS3 User Interface Module (Basic)**.
Herausgeber
Tantek Çelik
Website
www.w3.org/TR/css3-userint

CSS3 User Interface Module (Basic)

Status
W3C Candidate Recommendation, 11. Mai 2004
Beschreibung
Behandelt Pseudoklassen, Properties und Werte mit Bezug zur Darstellung des User Interfaces; erweitert entsprechende Elemente aus CSS 2.
Herausgeber
Tantek Çelik
Website
www.w3.org/TR/css3-ui/

CSS3 Values and Units

Status
W3C Working Draft, 19. September 2006
Beschreibung
Behandelt Einheiten und Werte, die in CSS verwendbar sind, sowie die Umsetzung angegebener in tatsächliche verwendete Werte; dieses Modul wird von den anderen CSS3-Modulen als Bezug verwendet.
Herausgeber
Håkon Wium Lie, Chris Lilley
Website
www.w3.org/TR/css3-values/

CSS3 Web Fonts Module

Status
W3C Working Draft, 2. August 2002
Beschreibung
Fontspezifische Properties und Werte; entspricht weitestgehend den entsprechenden Abschnitten aus CSS 2.
Herausgeber
Michel Suignard, Chris Lilley
Website
www.w3.org/TR/css3-webfonts

A.1.6 Weitere CSS-Ressourcen

Accessibility Features of CSS

Status
W3C Note, 4. August 1999
Beschreibung
Zusammenfassung der Features von CSS2, die unmittelbar Aspekte der Accessibility betreffen.
Herausgeber
Ian Jacobs, Judy Brewer
Website
www.w3.org/TR/CSS-access

CSS Mobile Profile 1.0

Status
W3C Candidate Recommendation, 25. Juli 2002
Beschreibung
Definiert eine Untergruppe von CSS Level 2, die auf die Beschränkungen und Erfordernisse von mobilen Endgeräten zugeschnitten ist.
Herausgeber
Ted Wugofski, Doug Dominiak, Peter Stark, Tapas Roy
Website
www.w3.org/TR/css-mobile

CSS Print Profile 1.0

Status
W3C Working Draft, 13. Oktober 2006
Beschreibung
Definiert eine Untergruppe von CSS Level 2.1 und CSS3 Paged Media Module, die, ausgehend von mobilen Endgeräten oder solchen ohne Druckertreiber, für Ausdrucke auf Low-cost-Druckern zugeschnitten ist; steht in Verbindung mit der Spezifikation »XHTML-Print«.
Herausgeber
Jim Bigelow
Website
www.w3.org/TR/css-print

A.1.7 W3C Ressourcen zu XML

Extensible Markup Language XML 1.0 (Fourth Edition)

Status
W3C Recommendation, 16. August 2006, editiert 29. September 2006
Beschreibung
Aktueller Standard. Beschreibt XML als Subset von SGML. Die vierte Version der Spezifikation berücksichtigt Errata und erforderliche Anpassungen, stellt dementsprechend also keine Neufassung gegenüber den Vorgängerversionen dar.
Herausgeber
Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau
Website
www.w3.org/TR/REC-xml

Extensible Markup Language XML 1.1 (Second Edition)

Status
W3C Recommendation, 16. August 2006, editiert 29. September 2006
Beschreibung
Aktueller Standard der Version 1.1. Beschreibt XML als Subset von SGML; berücksichtigt gegenüber XML 1.0 vor allem Änderungen im Unicode-Bereich; existiert parallel zu XML 1.0; Dokumente werden anhand der Versionsnummer unterschieden. Die zweite Version der Spezifikation berücksichtigt Errata und erforderliche Anpassungen.
Herausgeber
Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, John Cowan
Website
www.w3.org/TR/xml11/

XML Base

Status
W3C Recommendation, 27. Juni 2001
Beschreibung
Beschreibt die Definition von Basis-URIs für XML-Dokumente analog zu HTML Base.
Herausgeber
Jonathan Marsh
Website
www.w3.org/TR/xmlbase/

XML Events 2

Status
W3C Working Draft, 16. Februar 2007
Beschreibung
Geplante Nachfolgespezifikation zu »XML Events«; beschreibt das Hinzufügen von Eventlistenern und -handlern zu XML-Dokumenten (also auch XHTML-Dokumenten) gemäß DOM Level 2 auf interoperatible Weise.
Herausgeber
Shane McCarron, Steven Pemberton, T. V. Raman, Mark Birbeck
Website
www.w3.org/TR/xml-events

XML Events

Status
W3C Recommendation, 14. Oktober 2003
Beschreibung
Ehemals als »XHTML Events« bezeichnet.. Beschreibt das Hinzufügen von Eventlistenern und -handlern zu XML-Dokumenten (also auch XHTML-Dokumenten) gemäß DOM Level 2 auf interoperatible Weise.
Herausgeber
Shane McCarron, Steven Pemberton, T. V. Raman
Website
www.w3.org/TR/2003/REC-xml-events-20031014

XML Namespaces

Status
W3C Recommendation, 14. Januar 1999
Beschreibung
Methoden der Erzeugung so genannter qualifizierter Bezeichner in XML-Dokumenten durch Assoziation mit Namensräumen in Form von URI-Referenzen.
Herausgeber
Tim Bray, Dave Hollander, Andrew Layman
Website
www.w3.org/TR/REC-xml-names

XML Namespaces 1.1 (Second Edition)

Status
W3C Recommendation, 16. August 2006
Beschreibung
Methoden der Erzeugung so genannter qualifizierter Bezeichner in XML-Dokumenten durch Assoziation mit Namensräumen in Form von IRI-Referenzen (Internationalized Resource Identifier; zugehörige Beschreibung noch im Draftzustand: http://www.w3.org/International/iri-edit/).
Herausgeber
Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin
Website
www.w3.org/TR/xml-names11/

xml:id Version 1.0

Status
W3C Recommendation, 9. September 2005
Beschreibung
Die Bedeutung des Attributs `xml:id` in XML-Dokumenten und Methoden, dieses Attribut ohne Validierung und Verwendung externer Ressourcen einzusetzen.
Herausgeber
Jonathan Marsh, Daniel Veillard, Norman Walsh
Website
www.w3.org/TR/xml-id/

A.1.8 W3C-Ressourcen zu SVG

Scalable Vector Graphics SVG 1.1

Status
W3C Recommendation, 14. Januar 2003
Beschreibung
Aktueller Standard. Syntax und Eigenschaften von der Scalable Vector Graphics (SVG) Version 1.1, einer modularisierten, auf XML basierenden Sprache zur Beschreibung zweidimensionaler Vektor- und Vektor-/Rastergrafiken. **Ablösung der Spezifikation SVG 1.0**.
Herausgeber
Jon Ferraiolo, Jun Fujisawa, Dean Jackson
Website
www.w3.org/TR/SVG11/

Scalable Vector Graphics SVG 1.2

Status
W3C Working Draft, 13. April 2005
Beschreibung
Syntax und Eigenschaften von der SVG 1.2, einer modularisierten, auf XML basierenden Sprache zur Beschreibung animierter, interaktiver zweidimensionaler Vektor- und Vektor-/Rastergrafiken.
Herausgeber
Dean Jackson, Craig Northway
Website
www.w3.org/TR/SVG11/

Scalable Vector Graphics SVG Mobile

Status
W3C Recommendation, 14. Januar 2003
Beschreibung
Beschreibt auf der Grundlage von SVG 1.1 zwei Mobil-Profile von SVG – SVG Tiny für Mobiltelefone und SVG Basic für PDAs.
Herausgeber
Tolga Capin
Website
www.w3.org/TR/SVGMobile/

Scalable Vector Graphics SVG Tiny 1.2

Status
W3C Working Draft, 7. Dezember 2005
Beschreibung
Basisprofil von SVG, modularisierte 2D-Vektorsprache auf XML-Basis, verwendbar zur Grafikgenerierung für mobile Endgeräte.
Herausgeber
Ola Andersson, Robin Berjon, Jon Ferraiolo, Vincent Hardy, Scott Hayman, Dean Jackson, Chris Lilley, Craig Northway, Antoine Quint
Website
www.w3.org/TR/SVGMobile12/

A.1.9 W3C Ressourcen zu Accessibility

W3C Web Content Accessibility Guidelines 2.0

Status
W3C Working Draft, 23. November 2005
Beschreibung
Prinzipien, Richtlinien, Vorteile und Erfolgskriterien barrierefreier Information und Anwendungen; Anwendungsbeispiele.
Herausgeber
Ben Caldwell, Wendy Chisholm, J. Slatin, G. Vanderheiden
Website
www.w3.org/TR/WCAG20/

W3C Web Content Accessibility Guidelines 1.0

Status
W3C Recommendation, 5. Mai 1999
Beschreibung
Aktueller Standard. Enthält Richtlinien, nach denen Websites für Nutzer mit verschiedenen Behinderungen zugänglich gemacht werden können. Es werden hierbei verschiedene Level (Prioritäten) unterschieden. Die Spezifikation richtet sich sowohl an Web- wie an Applikationsentwickler und ist in verschiedenen Sprachen erhältlich.
Herausgeber
Gregg Vanderheiden, John Slatin und weitere
Website
www.w3.org/TR/WAI-WEBCONTENT Deutsche Übersetzung (noch nicht offiziell verifiziert): www.w3.org/Consortium/Offices/Germany/Trans/WAI/webinhalt.html

Core Techniques for Web Content Accessibility Guidelines 1.0

Status
W3C Note, 6. November 2000
Beschreibung
Grundlegende Vorgehensweisen zur Erstellung zugänglicher Inhalte entsprechend der Web Content Accessibility Guidelines 1.0.
Herausgeber
Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs
Website
www.w3.org/TR/2000/NOTE-WCAG10-CORE-TECHS/

HTML Techniques for Web Content Accessibility Guidelines 1.0

Status
W3C Note, 20. September 2000
Beschreibung
HTML-Techniken zur Erstellung barrierefreier Webseiten entsprechend der Web Content Accessibility Guidelines 1.0.
Herausgeber
Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs
Website
www.w3.org/TR/2000/NOTE-WCAG10-HTML-TECHS/

CSS Techniques for Web Content Accessibility Guidelines 1.0

Status
W3C Note, 20 September 2000
Beschreibung
CSS-Techniken zur Erstellung barrierefreier Inhalte entsprechend der Web Content Accessibility Guidelines 1.0.
Herausgeber
Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs
Website
www.w3.org/TR/WCAG10-CSS-TECHS/

A.1.10 Weitere W3C-Ressourcen

XForms 1.0 (Second Edition)

Status
W3C Recommendation, 14. März 2006
Beschreibung
XML-Anwendung; als Teil von XHTML konzipiert; beschreibt die kommende Generation von Webformularen mit stärkerer Trennung von Präsentation und Inhalt.
Herausgeber
John M. Boyer, David Landwehr, Roland Merrick, T. V. Raman, Micah Dubinko, Leigh L. Klotz Jr.
Website
www.w3.org/TR/xforms/

XForms 1.1

Status
W3C Working Draft, 3. November 2006
Beschreibung
Eine Erweiterung der bestehenden XForms 1.0 Spezifikation, die mehr Möglichkeiten der Datenmanipulation, der Datenübertragung und einen vergrößerten Set von Aktionen, Datentypen und Funktionen bietet.
Herausgeber
John M. Boyer
Website
www.w3.org/TR/xforms11/

XForms 1.0 Basic Profile

Status
W3C Candidate Recommendation, 14. Oktober 2003
Beschreibung
Eine Untergruppe von XForms, die für weniger leistungsfähige User Agents geeignet ist (mobile web).
Herausgeber
Micah Dubinko, T. V. Raman
Website
www.w3.org/TR/xforms-basic/

XFrames

Status
W3C Working Draft, 12. Oktober 2005
Beschreibung
XFrames ist eine XML-Anwendung die in Zusammenhang mit XHTML (1.1) die bisherigen HTML-Frames ablösen soll. Berücksichtigt werden die Frameunterstützung von User Agents und Bookmarkprobleme.
Herausgeber
Steven Pemberton, Masayasu Ishikawa
Website
www.w3.org/TR/xframes/

Ruby Annotation

Status
W3C Recommendation, 31. Mai 2001
Beschreibung
Erläuterung des Ruby-Markups für mehrzeiligen oder annotierten Text; vorwiegend für ostasiatischen Sprachraum von Bedeutung.
Herausgeber
Marcin Sawicki, M. Suignard, M. Ishikawa, M. Dürst, Tex Texin
Website
www.w3.org/TR/ruby/

Speech Synthesis Markup Language (SSML) Version 1.0

Status
W3C Recommendation, 7. September 2004
Beschreibung
Markupsprache auf XML-Basis zur Beschreibung der Spracherzeugung im Internet oder für andere Anwendungen (Bestandteil der Voice Browser Activity, siehe: http://www.w3.org/Voice/Activity.html)
Herausgeber
Daniel C. Burnett, Mark R. Walker, Andrew Hunt
Website
www.w3.org/TR/speech-synthesis/

Dieses Kapitel listet eine Reihe interessanter Onlineressourcen zu verschiedenen Aspekten auf, die dieses Buch behandelt. Ein Anspruch auf auch nur annähernde Vollständigkeit wird jedoch keinesfalls erhoben.

B Weitere Ressourcen

Auf den folgenden Seiten finden Sie ein Verzeichnis weiterer Websites, die Ihnen weiterführende Informationen zu den Themen diese Buchs bieten, aber auch Links zu Validatoren, Programmen und weiteren Tools, die für Sie als Webdesigner von Interesse ein können.

B.1 WHAT WG Initiative

Die WHAT WG (Web Hypertext Application Technology Working Group) ist eine Arbeitsgruppe namhafter Mitarbeiter verschiedener Browser- und Softwarefirmen (Apple, Mozilla, Opera), die sich die Weiterentwicklung verschiedener aktueller Webstandards zur Aufgabe gemacht hat.

Mitglieder der Arbeitsgruppe sind derzeit Brendan Eich, David Baron, David Hyatt, Dean Edwards, Håkon Wium Lie, Ian Hickson, Johnny Stenback und Maciej Stachowiak, die größtenteils auch im Rahmen des W3C tätig sind.

Web Application 1.0

Status
Working Draft, 3. März 2006
Beschreibung
Erweitert HTML und das DOM-Interface um die Erstellung webbasierter Anwendungen zu erleichtern.
Herausgeber
Ian Hickson
Website
www.whatwg.org/specs/web-apps/current-work/

Web Forms 2.0

Status
Working Draft, 3. März 2006
Beschreibung
Erweiterung der Formulare von HTML 4.0 und des entsprechenden DOM-Interfaces; richtet sich gleichermaßen an HTML- wie XHTML User Agents (soll in Zukunft in Web Applications integriert werden).
Herausgeber
Ian Hickson
Website
www.whatwg.org/specs/web-forms/current-work/

Web Controls 1.0

Status
Working Draft, 8. November 2004
Beschreibung
Konzipierte Erweiterungen zu CSS und zum DOM (sehr stark „work in progress“).
Herausgeber
Ian Hickson
Website
www.whatwg.org/specs/web-controls/current-work/

B.2 Mobiles Web

Openwave Developer Network

Beschreibung
Das Developer Network von Openwave stellt neben einer Vielzahl von Dokumentationen auch SDKs zur Verfügung, die Entwicklung mobiler Anwendungen auf der Basis von XHTML etc. ermöglichen, wie das Openwave Mobile SDK, Phone Simulator V7, MMS SDK und andere; Registrierung erforderlich.
Website
http://developer.openwave.com/

Open Mobile Alliance

Beschreibung
Die **Open Mobile Alliance** (OMA) stellt einen Zusammenschluss von derzeit rund 300 Unternehmen dar, die Dienstleistungen oder Technik für das mobile Web zur Verfügung stellen oder entwickeln. Integriert ist auch das ehemalige **Wap Forum** (www.wapforum.org). Zur Verfügung gestellt werden technische Informationen, White Papers und Spezifikationen; teilweise öffentlich; für den Mitgliederbereich Registrierung erforderlich.
Website
www.openmobilealliance.org

Nokia Developer Resources

Beschreibung
Auch **Nokia** stellt eine große Anzahl interessanter Tools, eigener und anderer SDKs (letztere per Link) und Dokumentationen für die Entwicklergemeinde im Mobilbereich zur Verfügung (wie Carbide, NetBeans Mobility Pack, CodeWarrior Development Studio, Borland JBuilder MobileSet und andere).
Website
www.forum.nokia.com

B.3 Browser und Browserspezifikationen

Opera

Website
Website des Opera-Browsers mit Downloadmöglichkeit und Informationen
www.opera.com

Mozilla/Firefox/SeaMonkey/Camino

Website
Website der Mozilla-Foundation mit Downloadmöglichkeit und Informationen zu allen Projekten.
www.mozilla.org

Internet Explorer

Website
Deutsche Downloadseite für den aktuellen Internet Explorer
www.microsoft.com/windows/ie_intl/de/download/

Evolt.org Browserarchiv

Beschreibung
Archiv mit (beinahe) allen historischen Browsern in verschiedenen Versionen für alle Betriebssystemplattformen. Hier gibt es auch Standalone-Versionen alter Internet Explorer, die sich parallel zum installierten Internet Explorer betreiben lassen. Nebenbei ist www.evolt.org auch ein sehr interessantes Magazin zu vielen Webdesignthemen.
Website
http://browsers.evolt.org/

Lynx (Textbrowser)

Beschreibung
Der Lynx ist auch heutzutage noch für verschiedene Plattformen ohne Weiteres erhältlich. Es gibt Versionen für Linux, Unix, VMS, DOS, Windows (95/98/NT) – inoffiziell auch unter Windows 2000/XP.
Plattform
Windows, DOS, Linux, Mac OS
Status
Open-Source-Software
Beurteilung
Das Original unter den Textbrowsern
Downloadgröße/Festplattenbedarf
ca. 0,7 MB/1,5 MB (je nach Plattform)
Website
http://lynx.isc.org
Lynx für Unix/Linux:
ftp://ftp2.cc.ukans.edu:80/pub/lynx/
Lynx für Windows:
http://www.jim.spath.com/lynx_win32/
Lynx für Windows/DOS:
http://www.fdisk.com/doslynx/lynxport.htm
Lynx für Mac OS:
http://www.lirmm.fr/~gutkneco/maclynx/

B.4 Webdeveloper-Tools

B.4.1 Validatoren

W3C XHTML-Validator

Beschreibung
Online-Service des W3C zum Validieren von Dokumenten. Dies kann durch Übergabe eines URL geschehen oder durch Dateiupload. Alternativ kann auch zu validierender Quellcode (vollständige Dokumente) mittels Texteingabefeld übergeben werden.
Website
http://validator.w3.org/

WDG HTML Validator

Beschreibung
Online-Validierungsservice der Web Design Group von Liam Quinn. Die Möglichkeiten entsprechen denjenigen des W3C-Validators.
Website
www.htmlhelp.com/tools/validator/

W3C CSS-Validator

Beschreibung
Online-Service des W3C zum Validieren von CSS-Dateien. Dies kann wie beim XHTML-Validator durch Übergabe eines URL geschehen oder durch Dateiupload. Alternativ kann auch zu validierender Quellcode (vollständige Stylesheets) mittels Texteingabefeld übergeben werden.
Website
http://jigsaw.w3.org/css-validator/

W3C Feed Validator

Beschreibung
Validiert Atom- oder RSS-Feeds (etwas neben dem Thema dieses Buches, aber praktisch)
Website
http://validator.w3.org/feed/

W3C Link Checker

Beschreibung
Prüft von einem Dokument ausgehende Verweise und Links auf „broken links“. Hierbei werden nicht nur `href`-, sondern auch `src`-Verweise getestet. Der Link Checker moniert nicht nur unerreichbare HTML-Dokumente, sondern auch Stylesheet-Links, Verweise auf Grafiken oder aus Gründen der Barrierefreiheit problematische JavaScript-Links.
Website
http://validator.w3.org/checklink/

W3C HTML Tidy

Beschreibung
Ein Online-Webinterface, mit dem sich Dokumente durch HTML Tidy bearbeiten lassen. Zur Verfügung gestellt von Tobias Nissen und Andreas Borutta.
Website
http://htmltidy.movb.de/?lang=de

Validome HTML / XHTML / WML / XML Validator

Beschreibung
Ermöglicht ähnlich wie bei den W3C-Validatoren die Online-Prüfung von Webseiten per URL-Angabe (Interface englisch, deutsch, französisch, russisch)
Website
www.validome.org

B.4.2 Toolbars

Webdeveloper-Toolbar

Beschreibung
Für Webdeveloper und Webdesigner unverzichtbare Browsertoolbar, mit der sich CSS und JavaScript der betrachteten HTML-Seite gezielt betrachten, aktivieren und deaktivieren lassen und der Seitenaufbau analysiert werden kann. Download für Firefox, Mozilla sowie Flock (Mac OS).
Autor
Chris Pederick
Website
http://chrispederick.com/work/webdeveloper/

B.4.3 Accessibility Tools

WEB for ALL – Accessibility-Toolbar

Beschreibung
Die **Web Accessibility Toolbar** wurde entwickelt, um die nicht-automatische Überprüfung von Webseiten hinsichtlich verschiedener Aspekte der Barrierefreiheit zu unterstützen. Sie wird vom AIS-Team (Accessible Information Solutions – Barrierefreie Informationslösungen) des Nationalen Informations- und Bibliotheksdienstes (NILS), Australien, zur Verfügung gestellt. Die Web Accessibility Toolbar ist derzeit nur mit dem Internet Explorer unter Windows nutzbar.
Sie können sie als Freeware bei **WEB for ALL** in deutschsprachiger Version unter folgender Webadresse beziehen (es stehen auch Versionen auf Spanisch, Französisch, Japanisch und Englisch zur Verfügung):
Website
www.webforall.info/html/deutsch/aistoolbar.php

Colorblind Web Page Filter

Beschreibung
Onlinetest. Simuliert die Wirkung verschiedener Arten von Farbenblindheit (Rot/Grün, Blau/Gelb und weitere) auf die Ansicht der angegebenen Webseite (beliebige URL-Angabe); filtert auch Grafiken entsprechend.
Autor
Matthew S. Wickline
Website
http://colorfilter.wickline.org/

Einfach für Alle – Styleswitcher und Accessibility-CSS

Beschreibung
Einfach für Alle bietet als praktische Zip- bzw. Sit-Archive eine Kollektion von JavaScript und CSS-Dateien an, die dem Webdeveloper die Erstellung barrierefreier Websites erleichtern. Neben einer leicht modifizierten Version des **Styleswitchers** von *A List Apart* findet sich auch ein Script zur **Schriftgrößenmanipulation**. Die vom User vorgenommenen Einstellungen werden durch Cookies gespeichert.
Website
www.einfach-fuer-alle.de/download/

B.5 Foren und Webzines

XHTML Forum

Beschreibung
Deutschsprachiges Forum von Dirk Henn (nicht nur) über XHTML. Themen sind ebenfalls CSS, Barrierefreiheit oder Serveradministration.
Website
www.xhtmlforum.de

A List Apart

Beschreibung
Webzine herausgegeben von Erin Kissane. Zahlreiche, durch namhafte Autoren (u.a. Eric Meyer, Jeffrey Zeldman, Molly E. Holzschlag) geschriebene, höchst interessante und innovative Artikel rund um Webdesign, CSS und Scripting (englisch).
Website
www.alistapart.com

SelfHTML

Beschreibung
Essentiell. Autor: Stefan Münz. *Das* deutschsprachige Tutorial über HTML, CSS, JavaScript, Perl, PHP und viele weitere Themen bezüglich Webdesign und Internettechnologien. Aktuelle Version 8.1.1 auch zum Download.
Website
http://de.selfhtml.org/

CSS Zen Garden

Beschreibung
Ständig aktualisierte Demonstration der verschiedenen, mit CSS möglichen Layouts, die sämtlich vom gleichen HTML-Gerüst ausgehen. Höchst lehrreich.
Website
www.csszengarden.com auch in deutscher Übersetzung: www.csszengarden.com/tr/deutsch/

B.6 Editoren

B.6.1 Quelltexteditoren

Ultra Edit

Beschreibung
Der Klassiker unter den Quelltexteditoren. Nicht ganz umsonst. Schweizer Messer unter den Editoren.
Website
www.ultraedit.com

Edit4Win

Beschreibung
Sehr guter Shareware-Editor für Windows mit deutschsprachiger Oberfläche. Auch für Programmierer geeignet.
Website
www.edit4win.de

Notepad++

Beschreibung
Fein aufgebohrter, auf Scintilla aufsetzender Notepad-Ersatz (aktuelle Version v4.0.2). Freeware. Empfehlenswert für HTML und weitere Sprachen.
Website
http://notepad-plus.sourceforge.net/de/site.htm

Arachnophilia

Beschreibung
Der gute, beinahe klassische javabasierte Webeditor von Paul Lutus. Freeware. Website beinhaltet gute Dokumentationen, nicht nur über das Programm.
Website
www.arachnoid.com/arachnophilia/

Weaverslave

Beschreibung
Sehr schöner Webeditor für Windows von Thomas Weinert. Freeware.
Website
www.weaverslave.ws

Bluefish

Beschreibung
Vielseitiger und vor allem vielsprachiger Webeditor von Olivier Sessink für Linux, Posix, Mac OS X. Vielversprechend (und -haltend).
Website
http://bluefish.openoffice.nl/

Pagespinner

Beschreibung
Kommerzieller, aber preisgünstiger Webeditor für Mac OS X. Unterstützt Apple Script. 21-Tage Testversion.
Website
www.optima-system.com

Quanta+

Beschreibung
Webeditor für Linux und Posix.
Website
http://sourceforge.net/projects/quanta/

Screem

Beschreibung
Quelltexteditor für Linux (GNOME)
Website
http://sourceforge.net/projects/screem/

Crimson Editor

Beschreibung
Sehr schöner, kompakter Freeware-Editor für Windows. Ingyu Kangs **Crimson Editor** eignet sich hervorragend als Notepad-Ersatz und besitzt alle benötigten Features.
Website
www.crimsoneditor.com

Phase 5 HTML-Editor

Beschreibung
Eindeutiger auf HTML und Webanwendungen festgelegt ist **Phase 5** für Windows von Hans-Dieter Berretz. Die für Privatanwender und Schulen freie Software kann mit kommerziellen professionellen Umgebungen sehr gut mithalten.
Website
www.ftp-uploader.de

B.6.2 WYSISWYG-HTML-Editoren

Adobe Dreamweaver (ex Macromedia)

Beschreibung
Kommerziell. Verbessert gegenüber der Vorgängerversion wurden die CSS-Bearbeitungsmöglichkeiten sowie die **CSS-Darstellung** in der Bearbeitungsvorschau, die unter der Bezeichnung »CSS Layout Visualization« zusätzlichen Komfort bietet (Windows 2000, XP, Mac OS X 10.3, 10.4).
Website
www.adobe.com/products/dreamweaver/

Adobe GoLive CS2

Beschreibung
Kommerziell. Die Philosophie hinter dem Programm ist vergleichbar mit Dreamweaver. Ein interessantes Feature könnte das Authoring für mobile Endgeräte (Palm-Handhelds, Pocket PCs und SmartPhones) sein, die bei Go Live stärker im Fokus sind (Windows, 2000, XP, Mac OS X).
Website
www.adobe.com/products/golive/

Nvu

Beschreibung
WYSIWYG-Editor aus der Open-Source-Gemeinde, der auf der Mozilla-Plattform und deren Gecko-Layoutengine basiert. Steht neben einer Version für Windows auch für Linux und Mac OS zur Verfügung.
Website
www.nvu-composer.de

C Inhalt der Begleit-CD

Die Begleit-CD enthält die vollständigen Quellcodebeispiele zu den Einführungs- und Referenzkapiteln sowie eine Reihe von nützlichen Tools und Editoren. Die CD bietet Ihnen zusätzlich eine HTML-Oberfläche mit Linklisten zu Referenzen und Downloadmöglichkeiten weiterer Tools.

C.1 Quellcodebeispiele – Einführungskapitel

Sie finden die Beispiele zu den Kapiteln 1 bis 13 auf der Begleit-CD im Verzeichnis `beispiele` und dort in den untergeordneten Verzeichnissen `kapitel_01` bis `kapitel_13`. Enthalten sind neben den HTML-Dateien auch CSS-Dateien und Grafiken.

C.2 Quellcodebeispiele – Referenzkapitel HTML

Sie finden die Quellcodebeispiele zu Kapitel 14, *(X)HTML Elementreferenz*, im Verzeichnis `beispiele/elemente`.

C.3 Tools und Werkzeuge

Im Ordner `werkzeuge` stehen Ihnen die im Rahmen des Buchs beschriebenen Tools zu Verfügung. Für alle aus Gründen der Aktualität oder Rechtslage nicht enthaltenen Programme und Utilities sind Downloadlinks genannt.

C.3.1 Hilfsprogramme

Im Verzeichnis `werkzeuge/tools` befinden sich Toolbars und weitere Hilfsprogramme für den Design- und Programmieralltag.

C.3.2 Editoren

Im Verzeichnis `werkzeuge/editoren` finden Sie eine Kollektion aktueller Quelltext- und WYSIWYG-Editoren für XHTML und CSS.

Index

A

Accessibility → Barrierefreiheit
Ankercontainer → Hyperlinkcontainer
Ankerpunkt 143
Application Level Eventhandler 920
Attribut 38
  Attributname 39
  Attributwert 39
  Attributwert, Begrenzer 39, 51
  leeres 51
  leeres, Begrenzer 51
Attribute
  abbr, <td> 853
  abbr, <th> 876
  accept, <form> 582
  accept, <input> 657
  accept-charset, <form> 582
  accesskey, <a> 412
  accesskey, <area> 447
  accesskey, <button> 499
  accesskey, <input> 657
  accesskey, <label> 682
  accesskey, <legend> 686
  accesskey, <textarea> 864
  action, <form> 583
  action, <isindex> (IE) 676
  action, <isindex> (NN) 677
  align, <applet> 440
  align, <caption> 505
  align, <col> 516
  align, <colgroup> 522
  *align, * 541
  align, <embed> 561
  align, <frame> (IE) 597
  align, <h1> – <h6> 617
  align, <hr> 622
  align, <iframe> 637
  align, <img> 645
  align, <input> 658
  align, <legend> 686
  align, <object> 749
  align, <p> 773
  align, <table> 832
  align, <tbody> 847
  align, <td> 853
  align, <tfoot> 872
  align, <th> 877
  align, <thead> 888
  align, <tr> 895
  alink, <body> 482
  allowtransparency, <frame> (IE) 597
  alt, <applet> 441
  alt, <area> 448
  alt, <embed> 562
  alt, <img> 646
  alt, <input> 659
  application, <frame> (IE) 598
  application, <iframe> (IE) 640
  archive, <applet> 441
  archive, <object> 750
  autocomplete, <form> (IE) 586
  autocomplete, <input> (IE) 664
  autostart, <embed> 562
  axis, <td> 854
  axis, <th> 877
  background, <body> 483
  background, <table> (IE) 840
  background, <table> (NN) 842
  background, <td> (IE) 860
  background, <td> (NN) 862
  background, <th> (IE) 884
  background, <th> (NN) 886
  background, <tr> (IE) 898
  behavior, <marquee> (IE) 709
  bgcolor, <body> 483
  bgcolor, <col> (IE) 519
  bgcolor, <colgroup> (IE) 525
  bgcolor, <marquee> (IE) 709
  bgcolor, <table> 833
  bgcolor, <tbody> (IE) 848
  bgcolor, <td> 854
  bgcolor, <tfoot> (IE) 874
  bgcolor, <th> 878
  bgcolor, <thead> (IE) 890
  bgcolor, <tr> 896
  bgproperties, <body> (IE) 485
  border, <embed> 562
  border, <frameset> (NN) 609
  border, <iframe> (IE) 640
  border, <img> 647
  border, <object> 750
  border, <table> 833
  bordercolor, <frame> (IE) 599

bordercolor, <frame> (NN) 600
bordercolor, <frameset> (IE) 607
bordercolor, <frameset> (NN) 610
bordercolor, <iframe> (IE) 641
bordercolor, <table> (IE) 840
bordercolor, <table> (NN) 843
bordercolor, <td> (IE) 861
bordercolor, <th> (IE) 884
bordercolor, <tr> (IE) 898
bordercolordark, <table> (IE) 840
bordercolordark, <td> (IE) 861
bordercolordark, <th> (IE) 885
bordercolordark, <tr> (IE) 898
bordercolorlight, <table> (IE) 841
bordercolorlight, <td> (IE) 861
bordercolorlight, <th> (IE) 885
bordercolorlight, <tr> (IE) 899
bottommargin, <body> (IE) 485
cellpadding, <table> 834
cellspacing, <table> 835
char, <col> 517
char, <colgroup> 523
char, <tbody> 847
char, <td> 855
char, <tfoot> 872
char, <th> 878
char, <thead> 889
char, <tr> 897
charoff, <col> 517
charoff, <tbody> 848
charoff, <td> 855
charoff, <tfoot> 873
charoff, <th> 878
charoff, <thead> 889
charoff, <tr> 897
charset, <a> 413
charset, <link> 695
charset, <script> 795
checked, <input> 659
cite, <blockquote> 478
cite, <del> 531
cite, <ins> 672
cite, <q> 787
class (Core Collection) 916
*class, * 813
classid, <object> 750
*clear,
* 495
clear, <table> (IE) 841
code, <applet> 441
code, <embed> 562
code, <object> (IE) 754
codebase, <applet> 441
codebase, <embed> 563
codebase, <object> 750
codetype, <object> 751
color, <basefont> 464
color, <font> 575
color, <hr> (IE) 624
cols, (NN) 542
cols, <frameset> 605
cols, <pre> (NN) 784
cols, <table> (NN) 843
cols, <textarea> 865
colspan, <td> 855
colspan, <th> 879
compact, <dir> 537
compact, <dl> 550
compact, <menu> 716
*compact, * 761
*compact, * 907
content, <meta> 720
controls, <img> (IE) 650
coord, <a> 414
cti, <a> (DoCoMo) 422
data, <object> 751
datetime, <del> 531
datetime, <ins> 673
declare, <object> 751
defer, <script> 795
dir (I18N Collection) 916
dir, <bdo> 468
direction, <marquee> (IE) 710
disabled, <a> (IE) 421
disabled, <button> 499
disabled, <input> 659
disabled, <link> (IE) 700
disabled, <optgroup> 766
disabled, <option> 769
disabled, <select> 805
disabled, <style> (IE) 822
disabled, <textarea> 865
dynsrc, <img> (IE) 651
dynsrc, <input> (IE) 664
effect, <font> (WebTV) 577
email, <a> (DoCoMo) 423
enctype, <form> 583
event, <script> (IE) 797
Eventhandler 915
face, <basefont> 465
face, <font> 575
fontsize, <body> (WebTV) 488
font-weight, <font> (NN) 577

for, <label> 682
for, <script> (IE) 798
frame, <table> 835
frameborder, <embed> 563
frameborder, <frame> 593
frameborder, <frameset> (IE) 608
frameborder, <iframe> 638
framespacing, <frameset> (IE) 608
gutter, (NN) 543
headers, <td> 856
height, <applet> 442
height, <embed> 563
height, <frame> (IE) 599
height, <iframe> 638
height, <img> 647
height, <marquee> (IE) 710
height, <object> 752
height, <table> (IE) 841
height, <table> (NN) 843
height, <td> 857
height, <th> 879
hidden, <embed> 563
hidefocus, <a> (IE) 421
href, <a> 414
href, <area> 449
href, <base> 459
href, <link> 695
hreflang, <a> 415
hreflang, <link> 695
hspace, <applet> 442
hspace, <embed> 564
hspace, <iframe> (IE) 641
hspace, <img> 647
hspace, <marquee> (IE) 710
hspace, <object> 752
hspace, <table> (NN) 844
http-equiv, <meta> 721
id (Core Collection) 917
id, <base> (IE) 460
id, <script> (IE) 797
*id, * 813
ijam, <a> (DoCoMo) 423
ilet, <a> (DoCoMo) 423
invertborder, <hr> (WebTV) 625
irst, <a> (DoCoMo) 424
ismap, <img> 648
ismap, <input> 660
ista, <a> (DoCoMo) 424
istyle, <input> (DoCoMo) 666
istyle, <textarea> (DoCoMo) 868
iswf, <a> (DoCoMo) 424
kana, <a> (DoCoMo) 425
label, <optgroup> 767
label, <option> 769
lang (I18N Collection) 917
lang (Universalattribut) 61
language, <script> 796
leftmargin, <body> (IE) 485
link, <body> 484
longdesc, <frame> 594
longdesc, <iframe> 638
longdesc, <img> 648
loop, <embed> 564
loop, <img> (IE) 651
loop, <marquee> (IE) 710
loopdelay, <img> (IE) 652
lowsrc, <img> (IE) 652
lowsrc, <img> (NN) 653
lowsrc, <input> (IE) 664
marginheight, <body> (NN) 487
marginheight, <frame> 594
marginheight, <iframe> 638
marginwidth, <body> (NN) 487
marginwidth, <frame> 594
marginwidth, <iframe> 638
maxlength, <input> 660
mayscript, <applet> (NN) 444
media, <link> 695
media, <style> 820
method, <form> 584
methods, <a> 415
multiple, <select> 805
n, <nextid> 739
name, <a> 415
name, <applet> 442
name, <button> 500
name, <embed> 564
name, <form> 584
name, <frame> 595
name, <iframe> 639
name, <img> 649
name, <input> 660
name, <link> (IE) 701
name, <map> 707
name, <meta> 721
name, <object> 752
name, <param> 777
name, <select> 805
name, <textarea> 866
nohref, <area> 449
noresize, <frame> 595
noresize, <iframe> (IE) 641

nosave, <img> (NN) 653
noshade, <hr> 623
notab, <area> (IE) 452
notab, <input> (IE) 665
notab, <object> (IE) 755
nowrap, <body> (IE) 486
nowrap, <dd> (IE) 528
nowrap, <dt> (IE) 555
nowrap, <table> (IE) 841
nowrap, <td> 857
nowrap, <th> 880
object, <applet> 442
object, <object> (IE) 755
palette, <embed> 564
ping, <a> (WebApp) 426
playcount, <embed> 565
pluginspage, <embed> 565
pluginurl, <embed> 565
point-size, <font> (NN) 576
profile, <head> 620
prompt, <isindex> 676
readonly, <input> 661
readonly, <textarea> 866
rel, <a> 417
rel, <link> 697
rev, <a> 417
rev, <link> 699
rightmargin, <body> (IE) 486
rows, <frameset> 607
rows, <textarea> 866
rowspan, <td> 858
rowspan, <th> 880
rules, <table> 837
runat, <script> (IE) 798
scheme, <meta> 722
scope, <td> 859
scope, <th> 881
scroll, <body> (IE) 486
scrollamount, <marquee> (IE) 711
scrolldelay, <marquee> (IE) 711
scrolling, <frame> 596
scrolling, <iframe> 639
selected, <option> 770
shape, <a> 418
shape, <area> 450
size, <basefont> 465
size, <font> 575
size, <hr> 623
size, <input> 661
size, <select> 806
span, <col> 518
span, <colgroup> 523
src, <applet> (IE) 443
src, <embed> 565
src, <frame> 596
src, <iframe> 639
src, <img> 649
src, <input> 662
src, <script> 796
Standardattribute 915
standby, <object> 753
start, <img> (IE) 652
*start, * 761
style (Style Collection) 918
*style, * 814
summary, <table> 839
suppress, <a> (NN) 422
suppress, <img> (NN) 654
tabindex, <a> 418
tabindex, <area> 450
tabindex, <button> 500
tabindex, <input> 662
tabindex, <object> 753
tabindex, <select> 806
tabindex, <textarea> 867
target, <a> 419
target, <area> 451
target, <base> 459
target, <form> 585
target, <isindex> (NN) 677
target, <link> 700
telbook, <a> (DoCoMo) 425
text, <body> 484
title (Core Collection) 918
title, <a> 420
title, <head> (IE) 620
title, <html> (IE) 629
title, <img> 649
topmargin, <body> (IE) 487
transparency, <font> (WebTV) 578
truespeed, <marquee> (IE) 711
type, <a> 420
type, <button> 501
type, <dir> (IE) 537
type, <dir> (NN) 538
type, <embed> 566
type, <input> 662
*type, * 690
type, <link> 700
type, <menu> (IE) 717
type, <menu> (NN) 718
type, <object> 753

*type, * 762
type, <param> 777
type, <script> 796
type, <style> 822
*type, * 907
units, <embed> 566
urn, <a> 420
urn, <link> 700
usemap, <img> 649
usemap, <input> 663
usemap, <object> 754
utn, <a> (DoCoMo) 425
utn, <form> (DoCoMo) 586
valign, <caption> (IE) 505
valign, <col> 518
valign, <colgroup> 524
valign, <table> (IE) 842
valign, <tbody> 848
valign, <td> 859
valign, <tfoot> 873
valign, <th> 883
valign, <thead> 889
valign, <tr> 897
value, <button> 501
value, <input> 663
*value, * 691
value, <option> 770
value, <param> 777
valuetype, <param> 777
vcard_name, <input> (IE) 665
version, <html> 628
vlink, <body> 484
volume, <embed> 566
vspace, <applet> 443
vspace, <embed> 566
vspace, <iframe> (IE) 641
vspace, <img> 650
vspace, <marquee> (IE) 712
vspace, <object> 754
vspace, <table> (NN) 844
width, <applet> 443
width, <col> 519
width, <colgroup> 525
width, (NN) 543
width, <embed> 566
width, <frame> (IE) 600
width, <hr> 623
width, <iframe> 640
width, <img> 650
width, <marquee> (IE) 712
width, <object> 754
width, <pre> 783
width, <table> 839
width, <td> 860
width, <th> 884
wrap, <pre> (NN) 784
wrap, <textarea> (NN) 868
wrap, <textarea>(IE) 867
xml:lang (I18N Collection) 919
xml:lang (Universalattribut) 61
xml:space, <pre> 783
xmlns, <html> 628
Attributsammlungen
Common 915
Core 915
I18N 915
Style 915
Außenabstand → CSS-Boxproperties, margin
Auszeichnungsbefehl → Marke
Auszeichnungskommando → Marke

B

Barrierefreie Informationstechnik-Verordnung (BITV) 241
Barrierefreiheit 239
Accessibility Tools 282
Accessibility-Attribute (IE) 947
Ausgabegerät 242
Barrierearmut 241
betroffene Gruppen 240, 241
Bilder 264
Bildschirmlupe 243
BITV 241
BITV-Selbstbewertung 245
Color Contrast Analyzer 248, 282
Datentabellen 251
Delorie Lynx Viewer 291
Downloadlink 268
Dreamweaver-Plug-in 244
Eventhandler 920
Eventhandler, geräteunabhängige 267
Events, korrespondierende 920
Farbkontraste 248
Formulare 257
Formulare, CSS-Gestaltung 262
Formulare, Eingabefelder 259
Formulare, Grundstruktur 258
Kurzcheck Bilder und Grafiken 265
Kurzcheck Datentabellen 253
Kurzcheck Download-Dateien 268

Kurzcheck dynamische Inhalte 268
Kurzcheck Farben 248
Kurzcheck Frames 247
Kurzcheck Imagemaps 266
Kurzcheck Layout 246
Kurzcheck Layouttabellen 247
Kurzcheck Links im Textinhalt 256
Kurzcheck Listen 250
Kurzcheck Markup von Inhalten 249
Kurzcheck Multimedia 266
Kurzcheck Navigation 255
Kurzcheck Popup-Fenster 267
Kurzcheck Schriften 248
Kurzcheck Scripte 267
Kurzcheck Seitenaufbau 246
Kurzcheck Sprache 250
Kurzcheck Verständlichkeit der Inhalte 269
Kurzcheck Weiterleitungen 267
Multimedia 264
Navigation 253
Navigation, Taststurbedienung 254
Nutzersicht 242
Regelungen, gesetzliche 241
Screenreader 243
Sitemap 256
Technologien 243
Textäquivalente 264
Vorteile 240
Wave Toolbar 281
Web Accessibility Initiative (WAI) 241
Web Accessibility Toolbar 248, 280, 291
Web Content Accessibility Guidelines (WCAG) 241
Yellowpipe Lynx Viewer 290
Basisschriftgröße 133
Basistextrichtung 916
Beispiel
HTML 4.01 frameset 80
HTML 4.01 strict 80
HTML 4.01 transitional 79
XHTML 1.0 frameset 82
XHTML 1.0 strict 81
XHTML 1.0 transitional 81
XHTML 1.1 82
XHTML 1.1 mit Schema-Locator 83
Bereichsnavigation 168
Bezeichner
lokaler 52
qualifizierter 52
Bildschirmlupe 243
Braille-Display 243
Breadcrumb-Navigation 169
Browser
Browserarchiv 1168
Browserarchiv, www.evolt.org 299
Camino 1167
Cello 299
Chimera 299
Evolt.org 1168
Firefox 288, 348, 359, 1167
Internet Explorer 1167
Lynx 289, 1168
Mosaic 299
Mozilla 1167
Safari 348
SeaMonkey 1167
Textbrowser-Simulationen 288
Viola 299
X-Smiles 359

C

Cascading-Prinzip 103, 999
Präferenz in der Präferenzkaskade 999
Spezifizität der Selektoren 1000
CDATA-Textinhalt 42
Character-Referenz 40
Checkboxen 157
closing delimiter 36
CMYK → Farben, Vierfarbmodell
CSS At-Rules 1003
@charset 1003
@font-face 1004
@import 1004
@media 1005
@page 1006
CSS für mobile Geräte 228
Abbildungen 230
Ankercontainer 230
Dokumentbereiche 229
Formulare 232
Listen 232
Margins 229
media= 228
Menüs 230
Navigation 230
Schriftfamilie 231
Skiplinks 230
Tabellen 231
Text 230
Wrapper-Container 229

CSS Mobile Profile 228
CSS-Boxmodell 105
Außenabstand 105
Box-Properties 105
Innenabstand 105
Innenbreite 105
Innenhöhe 105
Rahmeneigenschaften 105
CSS-Boxproperties
margin 1055
CSS-Formatierungsregeln 91, 991
!important-Schalter 993
Reihenfolge 94, 993
CSS-Kommentarsyntax 102, 1002
CSS-Maßeinheiten 1023
em 1023
ex 1026
Längenangaben 1023
Längenangaben, absolut 1023
Längenangaben, relativ 1023
Prozentangaben 1026
CSS-Properties 1028
background 1044
background-attachment 1045
background-color 1046
background-image 1047
background-position 1048
background-position, Längenangaben 1048
background-position, Prozentangaben 1048
background-position, Schlüsselwörter 1049
background-repeat 1045, 1050
border 1056
border-collapse 1103
border-color 1057
border-spacing 1104
border-style 1058
border-width 1061
bottom 1072
Boxproperties 1053
Boxproperties, Anwendbarkeit 1053
Boxproperties, Containerinhalt 1053
Boxproperties, Inhaltsbereich 1054
browserspezifische 1121
caption-side 1106
clear 1074
clip 1075
color 1052
content 1114
Contentgenerierung 1114
counter-increment 1115
counter-reset 1116
cursor 1111
direction 1028
display 1090
Display-Property 1090
empty-cells 1108
Farben 106
Farben und Hintergründe 1044
float 1076
font 1038
font-family 1038
Fontproperties 1038
font-size 1039
font-size-adjust 1040
font-stretch 1041
font-style 1041
font-variant 1042
font-weight 1043
font-weight, Fettegrad 1043
height 1063
Hintergrundeigenschaften 106
Kontur-Properties 1067
left 1079
letter-spacing 1029
line-height 1029
Listenstyle-Properties 1098
list-style 1098
list-style-image 1099
list-style-position 1100
list-style-type 1100
margin 1055
max-height 1065
max-width 1066
min-height 1067
min-width 1067
orphans 1120
outline 1067
outline-color 1068
outline-style 1069
outline-width 1071
overflow 1080
padding 1054
page 1117
page-break-after 1118
page-break-before 1119
page-break-inside 1119
position 1082
Positioning-Properties 1072
quotes 1116

right 1083
scrollbar-3dlight-color 1122
scrollbar-arrow-color 1122
scrollbar-base-color 1123
scrollbar-darkshadow-color 1124
scrollbar-face-color 1124
scrollbar-highlight-color 1125
Scrollbar-Properties 1121
scrollbar-shadow-color 1126
scrollbar-track-color 1126
Seitenformatierung 1117
Sonstige Properties 1111
Tabellen-Properties 1103
table-layout 1109
text-align 1030
text-decoration 1030
text-indent 1031
text-indent, hängender Einzug 1032
Textproperties 1028
text-shadow 1033
text-transform 1033
top 1084
unicode-bidi 1034
vertical-align 1035
visibility 1086
white-space 1036
widows 1120
width 1064
word-spacing 1037
z-index 1089
CSS-Pseudoelemente 1014
:after 1015
:before 1015
:first-letter 1016
:first-line 1016
CSS-Pseudoklassen 1017
:active 1018
:focus 1018
:hover 1019
:lang 1020
:link 1020
:visited 1021
CSS-Pseudo-Kontextselektoren 1021
:first-child 1021
CSS-Regel
Basis-URI 1047
CSS-Selektoren 92
Adjacent-Sibling-Selektor 1012
attributabhängige 1013
Child-Selektor 1011
Descendant Selektor 1011
Gruppenselektor 1008
Gruppierung 93
ID-Selektor 97, 1010
Klassenselektor 1008
Klassenselektor, frei 95
Klassenselektor, freier 1009
Klassenselektor, gebunden 96
Klassenselektor, gebundener 1009
Klassenselektor, Mehrfachzuweisung 1009
Klassenselektor, Wahl des Bezeichners 1010
Typselektor 93, 1007
Universeller Selektor 1007
Wahl des Selektors 101
CSS-Styledefinition
Dokumentstyle 88, 996
Inlinestyle 90, 995
Position 85, 995
Stylesheet, importiert 88, 997
Stylesheet, verlinkt 87, 998
CSS-Werte 1023
Farben 1027
Farbnamen 1027
Hexadezimalfarbangabe 1027
Hexadezimalfarbangabe, gekürzt 1027
RGB-Wert, numerisch 1027
RGB-Wert, prozentual 1028

D

Databinding-Attribute (IE) 949
Default-Namensraum 53
Definition
Attribute, Common Collection 915
Barrierefreiheit 61
Doctype-Switching 1127
Floaten 116
Hyperreferenz 87
Markup 22
MIME-Typ 58
Typgültigkeit 60
valid 60
Validator 60
Validierung 60
W3C Candidate Recommendation 1138
W3C Proposed Edited Recommendation 1138
W3C Proposed Recommendation 1138
W3C Recommendation 1138
W3C Working Draft 1138

W3C Working Group Note 1138
well-formed 60
Wohlgeformtheit 60
Definitionsliste 149
Design für mobile Geräte 222
Accesskeys 227
Bilder 225
Farben 226
Farbtiefe 226
Grafiken 225
Kontrast 226
Linearisierung 222
Markup-Sprache, Wahl 219
Navigation 227
Quelltextaufbau 224
Quelltextreihenfolge 223
Schriftgrößen 224
Tabellen 225
Test mit Openwave Phone Simulator 236
Test mit Opera 233
Test mit Opera, Normalansicht 233
Test mit Opera, SSR 235
Textauszeichnung 225
Textfarbe 227
Doctype-Deklaration 76
Aufbau 77
Name 77
PUBLIC 77
Public-Identifier 77
Public-Identifier, Aufbau 78
SYSTEM 77
System-Identifier 77
Dokumentinhalt
Gestaltung 131
Dokumentkopf 23
Dokumentrumpf 23
Dokumenttyp-Definition 43
Elementvereinbarung 43
Dokumenttyp-Deklaration 53
DTD-Variante 54

E

Editoren 1173
Adobe Dreamweaver 1175
Adobe GoLive CS2 1175
Arachnophilia 272, 1173
Bluefish 206, 272, 1174
Crimson Editor 272, 1174
Dreamweaver 274
Edit4Win 272, 1173
GoLive 274
Homesite 272
HTML-Kit 206, 210
jEdit 271
Namo Web Editor 274
Notepad 271
Notepad++ 272, 1173
Nvu 274, 275, 1175
Pagespinner 272, 1174
Phase 5 272, 273
Phase 5 HTML-Editor 1175
Quanta 206, 272
Quanta+ 1174
SciTE 271
Screem 272, 1174
SuperHTML 274
tsWebEditor 206
Ultra Edit 271, 1173
Weaverslave 272, 1173
XFormation 361
Element 22, 36
Abschluss 49
Container 38
Inhaltsmodell 37
leeres 38
Verkürzung 48
Verschachtelung 37
Elemente
<a> 142, 411
<a>, accesskey 145, 412
<a>, charset 413
<a>, coord 414
<a>, cti (DoCoMo) 422
<a>, disabled (IE) 421
<a>, email (DoCoMo) 423
<a>, hidefocus (IE) 421
<a>, href 145, 414
<a>, hreflang 415
<a>, ijam (DoCoMo) 423
<a>, ilet (DoCoMo) 423
<a>, irst (DoCoMo) 424
<a>, ista (DoCoMo) 424
<a>, iswf (DoCoMo) 424
<a>, kana (DoCoMo) 425
<a>, methods 415
<a>, name 143, 415
<a>, ping (WebApp) 426
<a>, rel 417
<a>, rev 417
<a>, shape 418
<a>, suppress (NN) 422

<a>, tabindex 145, 418
<a>, target 145, 419
<a>, telbook (DoCoMo) 425
<a>, title 145, 420
<a>, type 420
<a>, urn 420
<a>, utn (DoCoMo) 425
<abbr> 139, 430
<abbr>, title 139
<acronym> 139, 433
<acronym>, title 139
<address> 437
<applet> 439
<applet>, align 440
<applet>, alt 441
<applet>, archive 441
<applet>, code 441
<applet>, codebase 441
<applet>, height 442
<applet>, hspace 442
<applet>, mayscript (NN) 444
<applet>, name 442
<applet>, object 442
<applet>, src (IE) 443
<applet>, vspace 443
<applet>, width 443
<area> 446
<area>, accesskey 447
<area>, alt 448
<area>, href 449
<area>, nohref 449
<area>, notab (IE) 452
<area>, shape 450
<area>, tabindex 450
<area>, target 451
<b> 139, 140, 141, 454
<base> 69, 458
<base>, href 459
<base>, id (IE) 460
<base>, target 459
<basefont> 463
<basefont>, color 464
<basefont>, face 465
<basefont>, size 465
<bdo> 466
<bdo>, dir 468
<big> 141, 473
<blockquote> 72, 476
<blockquote>, cite 478
<body> 60, 481
<body>, alink 482
<body>, background 483
<body>, bgcolor 483
<body>, bgproperties (IE) 485
<body>, bottommargin (IE) 485
<body>, fontsize (WebTV) 488
<body>, Inhaltsmodell 69
<body>, leftmargin (IE) 485
<body>, link 484
<body>, marginheight (NN) 487
<body>, marginwidth (NN) 487
<body>, nowrap (IE) 486
<body>, rightmargin (IE) 486
<body>, scroll (IE) 486
<body>, text 484
<body>, topmargin (IE) 487
<body>, vlink 484
*
* 160, 493
*
, clear* 495
<button> 158, 498
<button>, accesskey 499
<button>, disabled 499
<button>, name 500
<button>, tabindex 500
<button>, type 501
<button>, value 501
<caption> 153, 503
<caption>, align 505
<caption>, valign (IE) 505
* * 507
<cite> 138, 510
<code> 138, 512
<col> 153, 515
<col>, align 516
<col>, bgcolor (IE) 519
<col>, char 517
<col>, charoff 517
<col>, span 518
<col>, valign 518
<col>, width 519
<colgroup> 153, 521
<colgroup>, align 522
<colgroup>, bgcolor (IE) 525
<colgroup>, charoff 523
<colgroup>, span 524
<colgroup>, valign 524
<colgroup>, width 525
<dd> 149, 527
<dd>, nowrap (IE) 528
<del> 74, 140, 530
<del>, cite 531
<del>, datetime 531

<*dfn*> 138, 533
<*dir*> 536
<*dir*>, *compact* 537
<*dir*>, *type (IE)* 537
<*dir*>, *type (NN)* 538
<*div*> 70, 539
<*div*>, *align* 541
<*div*>, *cols (NN)* 542
<*div*>, *Div-Layout* 71
<*div*>, *gutter (NN)* 543
<*div*>, *width (NN)* 543
<*dl*> 149, 549
<*dl*>, *compact* 550
<*dt*> 149, 554
<*dt*>, *nowrap (IE)* 555
<*em*> 138, 557
<*embed*> *(Prop.)* 560
<*embed*>, *align* 561
<*embed*>, *alt* 562
<*embed*>, *autostart* 562
<*embed*>, *border* 562
<*embed*>, *code* 562
<*embed*>, *codebase* 563
<*embed*>, *frameborder* 563
<*embed*>, *height* 563
<*embed*>, *hidden* 563
<*embed*>, *hspace* 564
<*embed*>, *loop* 564
<*embed*>, *name* 564
<*embed*>, *palette* 564
<*embed*>, *playcount* 565
<*embed*>, *pluginspage* 565
<*embed*>, *pluginurl* 565
<*embed*>, *src* 565
<*embed*>, *type* 566
<*embed*>, *units* 566
<*embed*>, *volume* 566
<*embed*>, *vspace* 566
<*embed*>, *width* 566
<*fieldset*> 160, 569
<*font*> 142, 573
<*font*>, *color* 575
<*font*>, *effect (WebTV)* 577
<*font*>, *face* 575
<*font*>, *font-weight (NN)* 577
<*font*>, *point-size (NN)* 576
<*font*>, *size* 575
<*font*>, *transparency (WebTV)* 578
<*form*> 73, 156, 580
<*form*>, *accept* 582
<*form*>, *accept-charset* 582
<*form*>, *action* 156, 583
<*form*>, *autocomplete (IE)* 586
<*form*>, *enctype* 158, 583
<*form*>, *method* 156, 584
<*form*>, *name* 584
<*form*>, *target* 585
<*form*>, *utn (DoCoMo)* 586
<*frame*> 63, 592
<*frame*>, *align (IE)* 597
<*frame*>, *allowtransparency (IE)* 597
<*frame*>, *application (IE)* 598
<*frame*>, *bordercolor (IE)* 599
<*frame*>, *bordercolor (NN)* 600
<*frame*>, *frameborder* 593
<*frame*>, *height (IE)* 599
<*frame*>, *longdesc* 594
<*frame*>, *marginheight* 594
<*frame*>, *marginwidth* 594
<*frame*>, *name* 595
<*frame*>, *noresize* 595
<*frame*>, *scrolling* 596
<*frame*>, *src* 596
<*frame*>, *width (IE)* 600
<*frameset*> 63, 604
<*frameset*>, *border (NN)* 609
<*frameset*>, *bordercolor (IE)* 607
<*frameset*>, *bordercolor (NN)* 610
<*frameset*>, *cols* 605
<*frameset*>, *frameborder (IE)* 608
<*frameset*>, *framespacing (IE)* 608
<*frameset*>, *rows* 607
<*h1*> 71
<*h1*> – <*h6*> 615
<*h1*> – <*h6*>, *align* 617
<*h1*>-<*h6*> 134
<*head*> 60, 618
<*head*>, *profile* 620
<*head*>, *title (IE)* 620
<*hr*> 72, 621
<*hr*>, *align* 622
<*hr*>, *color (IE)* 624
<*hr*>, *invertborder (WebTV)* 625
<*hr*>, *noshade* 72, 623
<*hr*>, *size* 623
<*hr*>, *width* 623
<*html*> 60, 626
<*html*>, *title (IE)* 629
<*html*>, *version* 628
<*html*>, *xmlns* 628
<*i*> 140, 141, 633
<*iframe*> 147, 635

<iframe>, align 637
<iframe>, application (IE) 640
<iframe>, border (IE) 640
<iframe>, bordercolor (IE) 641
<iframe>, frameborder 638
<iframe>, height 638
<iframe>, hspace (IE) 641
<iframe>, longdesc 638
<iframe>, marginheight 638
<iframe>, marginwidth 638
<iframe>, name 639
<iframe>, noresize (IE) 641
<iframe>, scrolling 639
<iframe>, src 639
<iframe>, vspace (IE) 641
<iframe>, width 640
<img> 146, 643
<img>, align 645
<img>, alt 646
<img>, border 148, 647
<img>, controls (IE) 650
<img>, dynsrc (IE) 651
<img>, height 146, 647
<img>, hspace 647
<img>, ismap 648
<img>, longdesc 648
<img>, loop (IE) 651
<img>, loopdelay (IE) 652
<img>, lowsrc (IE) 652
<img>, lowsrc (NN) 653
<img>, name 649
<img>, nosave (NN) 653
<img>, src 146, 649
<img>, start (IE) 652
<img>, suppress (NN) 654
<img>, title 649
<img>, usemap 649
<img>, vspace 650
<img>, width 146, 650
<input> 156, 655
<input>, accept 657
<input>, accesskey 157, 657
<input>, align 658
<input>, alt 659
<input>, autocomplete (IE) 664
<input>, checked 659
<input>, disabled 659
<input>, dynsrc (IE) 664
<input>, id 157
<input>, ismap 660
<input>, istyle (DoCoMo) 666
<input>, lowsrc (IE) 664
<input>, maxlength 660
<input>, name 157, 660
<input>, notab (IE) 665
<input>, readonly 661
<input>, size 661
<input>, src 662
<input>, tabindex 662
<input>, type 156, 662
<input>, usemap 663
<input>, value 157, 663
<input>, vcard_name (IE) 665
<ins> 74, 140, 671
<ins>, cite 672
<ins>, datetime 673
<isindex> 674
<isindex>, action (IE) 676
<isindex>, action (NN) 677
<isindex>, prompt 676
<isindex>, target (NN) 677
<kbd> 138, 679
<label> 157, 681
<label>, accesskey 682
<label>, for 682
<legend> 685
<legend>, accesskey 686
<legend>, align 686
<li> 688
<li>, type 690
<li>, value 691
<link> 67, 693
<link>, charset 695
<link>, disabled (IE) 700
<link>, href 695
<link>, hreflang 695
<link>, media 695
<link>, name (IE) 701
<link>, rel 697
<link>, rev 699
<link>, target 700
<link>, type 700
<link>, urn 700
<listing> 704
<map> 706
<map>, name 707
<marquee> (IE) 708
<marquee>, behavior (IE) 709
<marquee>, bgcolor (IE) 709
<marquee>, direction (IE) 710
<marquee>, height (IE) 710
<marquee>, hspace (IE) 710

<marquee>, loop (IE) 710
<marquee>, scrollamount (IE) 711
<marquee>, scrolldelay (IE) 711
<marquee>, truespeed (IE) 711
<marquee>, vspace (IE) 712
<marquee>, width (IE) 712
<menu> 715
<menu>, compact 716
<menu>, type (IE) 717
<menu>, type (NN) 718
<meta> 66, 719
<meta>, content 720
<meta>, http-equiv 721
<meta>, name 721
<meta>, scheme 722
<nextid> 738
<nextid>, n 739
<noembed> 568
<noframes> 63, 740
<noscript> 74, 744
<object> 69, 147, 748
<object>, align 749
<object>, archive 750
<object>, border 750
<object>, classid 750
<object>, code (IE) 754
<object>, codebase 750
<object>, codetype 751
<object>, data 751
<object>, declare 751
<object>, height 752
<object>, hspace 752
<object>, name 752
<object>, notab (IE) 755
<object>, object (IE) 755
<object>, standby 753
<object>, tabindex 753
<object>, type 753
<object>, usemap 754
<object>, vspace 754
<object>, width 754
<ol> 148, 150, 759
<ol>, compact 761
<ol>, start 151, 761
<ol>, type 151, 762
<optgroup> 159, 765
<optgroup>, disabled 766
<optgroup>, label 767
<option> 159, 768
<option>, disabled 159, 769
<option>, label 769
<option>, selected 770
<option>, value 770
<p> 71, 135, 772
<p>, align 773
<p>, Defaultmargins 135
<p>, Inhaltsmodell 71
<param> 69, 147, 775
<param>, name 777
<param>, type 777
<param>, value 777
<param>, valuetype 777
<plaintext> 779
<pre> 72, 781
<pre>, cols (NN) 784
<pre>, width 783
<pre>, wrap (NN) 784
<pre>, xml:space 783
<q> 138, 785
<q>, cite 787
<s> 142, 789
<samp> 138, 791
<script> 65, 68, 794
<script>, charset 795
<script>, defer 795
<script>, event (IE) 797
<script>, for (IE) 798
<script>, id (IE) 797
<script>, Inhaltsmodell 68
<script>, language 796
<script>, runat (IE) 798
<script>, src 796
<script>, type 796
<select> 159, 803
<select>, disabled 805
<select>, multiple 159, 805
<select>, name 159, 805
<select>, size 806
<select>, tabindex 806
<small> 141, 809
** 139, 812
, class 813
, id 813
, style 814
<strike> 142, 815
** 138, 817
<style> 65, 67, 819
<style>, disabled (IE) 822
<style>, Inhaltsmodell 67
<style>, media 820
<style>, type 822
<sub> 141, 826

<*sup*> 141, 828
<*table*> 73, 152, 830
<*table*>, *align* 832
<*table*>, *background (IE)* 840
<*table*>, *background (NN)* 842
<*table*>, *bgcolor* 833
<*table*>, *border* 833
<*table*>, *bordercolor (IE)* 840
<*table*>, *bordercolor (NN)* 843
<*table*>, *bordercolordark (IE)* 840
<*table*>, *bordercolorlight (IE)* 841
<*table*>, *cellpadding* 834
<*table*>, *cellspacing* 835
<*table*>, *clear (IE)* 841
<*table*>, *cols (NN)* 843
<*table*>, *frame* 835
<*table*>, *height (IE)* 841
<*table*>, *height (NN)* 843
<*table*>, *hspace (NN)* 844
<*table*>, *nowrap (IE)* 841
<*table*>, *rules* 837
<*table*>, *summary* 839
<*table*>, *valign (IE)* 842
<*table*>, *vspace (NN)* 844
<*table*>, *width* 839
<*tbody*> 845
<*tbody*>, *align* 847
<*tbody*>, *bgcolor (IE)* 848
<*tbody*>, *char* 847
<*tbody*>, *charoff* 848
<*tbody*>, *valign* 848
<*td*> 152, 851
<*td*>, *abbr* 853
<*td*>, *align* 853
<*td*>, *axis* 854
<*td*>, *background (IE)* 860
<*td*>, *background (NN)* 862
<*td*>, *bgcolor* 854
<*td*>, *bordercolor (IE)* 861
<*td*>, *bordercolordark (IE)* 861
<*td*>, *bordercolorlight (IE)* 861
<*td*>, *char* 855
<*td*>, *charoff* 855
<*td*>, *colspan* 855
<*td*>, *headers* 856
<*td*>, *height* 857
<*td*>, *nowrap* 857
<*td*>, *rowspan* 858
<*td*>, *valign* 859
<*td*>, *width* 860
<*textarea*> 158, 863
<*textarea*>, *accesskey* 864
<*textarea*>, *cols* 865
<*textarea*>, *disabled* 865
<*textarea*>, *istyle (DoCoMo)* 868
<*textarea*>, *name* 866
<*textarea*>, *readonly* 866
<*textarea*>, *rows* 866
<*textarea*>, *tabindex* 867
<*textarea*>, *wrap (IE)* 867
<*textarea*>, *wrap (NN)* 868
<*tfoot*> 871
<*tfoot*>, *align* 872
<*tfoot*>, *bgcolor (IE)* 874
<*tfoot*>, *char* 872
<*tfoot*>, *charoff* 873
<*tfoot*>, *valign* 873
<*th*> 152, 875
<*th*>, *abbr* 876
<*th*>, *align* 877
<*th*>, *axis* 877
<*th*>, *background (IE)* 884
<*th*>, *background (NN)* 886
<*th*>, *bgcolor* 878
<*th*>, *bordercolor (IE)* 884
<*th*>, *bordercolordark (IE)* 885
<*th*>, *bordercolorlight (IE)* 885
<*th*>, *char* 878
<*th*>, *charoff* 878
<*th*>, *colspan* 879
<*th*>, *height* 879
<*th*>, *nowrap* 880
<*th*>, *rowspan* 880
<*th*>, *scope* 881
<*th*>, *valign* 883
<*th*>, *width* 884
<*thead*> 887
<*thead*>, *align* 888
<*thead*>, *bgcolor (IE)* 890
<*thead*>, *char* 889
<*thead*>, *charoff* 889
<*thead*>, *valign* 889
<*title*> 60, 892
<*tr*> 152, 894
<*tr*>, *align* 895
<*tr*>, *background (IE)* 898
<*tr*>, *bgcolor* 896
<*tr*>, *bordercolor (IE)* 898
<*tr*>, *bordercolordark (IE)* 898
<*tr*>, *bordercolorlight (IE)* 899
<*tr*>, *char* 897
<*tr*>, *charoff* 897

<tr>, valign 897
<tt> 900
<u> 142, 903
<ul> 74, 148, 150, 905
<ul>, compact 907
<ul>, type 151, 907
<var> 138, 910
<xmp> 912
Strukturelemente 60
Elementname 44
Encoding
Escapesequenzen 971
Markierungszeichen 972
Trennzeichen 972
URL-Encoding 971
Zeichen, alphanumerische 972
Zeichen, nicht-druckbare 973
Zeichen, nicht-reservierte 972
Zeichen, reservierte 972
Zeichen, verbotene 973
Endmarke 22, 36
Endtag 22
Endtag → Endmarke
Entität 40
vordefinierte in SGML 40
Entities
Akzentzeichen 956
Buchstabenformen, internationale 961
Character-Entities 968
Großbuchstaben mit Akzentzeichen 957
Großbuchstaben, griechische 959
Interpunktionszeichen 961
Kleinbuchstaben mit Akzentzeichen 958
Kleinbuchstaben, griechische 960
Ligaturen 959
Pfeile 965
Spationierung 966
Steuerzeichen 955
Symbole in Buchstabenform 965
Symbole, höhere mathematische 962
Symbole, kommerzielle 956
Symbole, mathematische 962
Symbole, Währungen 966
Symbole, weitere 965
Symbole, Zahlen 967
Whitespace 966
Zahlen 967
Zeichen, typographische 963
Ersetzungsvorschrift → Entität
Eventhandler 915
Applikationsebene 920
geräteabhängige 920
geräteunabhängige 920
Interaktionsebene 920
onabort (IE) 928
onactivate (IE) 929
onafterprint (IE) 929
onafterupdate (IE) 929
onbeforeactivate (IE) 930
onbeforecopy (IE) 930
onbeforecut (IE) 930
onbeforedeactivate (IE) 931
onbeforeeditfocus (IE) 931
onbeforepaste (IE) 932
onbeforeprint (IE) 932
onbeforeunload (IE) 932
onbeforeupdate (IE) 933
onblur (Intrinsic) 921
onbounce (IE) 933
oncellchange (IE) 934
onchange (Intrinsic) 921
onclick (Common) 922
oncontextmenu (IE) 934
oncopy (IE) 934
oncut (IE) 934
ondataavailable (IE) 935
ondatasetchanged (IE) 935
ondatasetcomplete (IE) 935
ondblclick (Common) 923
ondeactivate (IE) 936
ondrag (IE) 936
ondragend (IE) 936
ondragenter (IE) 936
ondragleave (IE) 937
ondragover (IE) 937
ondragstart (IE) 937
ondrop (IE) 937
onerror (IE) 938
onerrorupdate (IE) 938
onfilterchange (IE) 938
onfinish (IE) 938
onfocus (Intrinsic) 923
onfocusin (IE) 939
onfocusout (IE) 939
onhelp (IE) 939
onkeydown (Common) 923
onkeypress (Common) 924
onkeyup (Common) 924
onlayoutcomplete (IE) 940
onload (Intrinsic) 924
onlosecapture (IE) 940
onmousedown (Common) 925

onmouseenter (IE) 940
onmouseleave (IE) 941
onmousemove (Common) 925
onmouseover(Common) 926
onmouseup (Common) 925, 926
onmousewheel (IE) 941
onmove (IE) 941
onmoveend (IE) 942
onmovestart (IE) 942
onpaste (IE) 942
onpropertychange (IE) 942
onreadystatechange (IE) 943
onreset (Intrinsic) 926
onresize (IE) 943
onresizeend (IE) 944
onresizestart (IE) 944
onrowenter (IE) 944
onrowexit (IE) 945
onrowsdelete (IE) 945
onrowsinserted (IE) 945
onscroll (IE) 946
onselect (Intrinsic) 927
onselectstart (IE) 946
onstart (IE) 946
onstop (IE) 947
onsubmit (Intrinsic) 927
onunload (Intrinsic) 927
Schreibweise 921
Events
äquivalente Events 920

F

Farben 1027
8-Bit Palette 981
browsersafe 981
CLUT-Palette 981
CSS3 Color Module 983
Differenz PC/Mac 982
Farbbezeichner 982
Farbbezeichner CSS3 988
Farbraum 982
Farbsynthese 981
Farbsynthese, additive 982
Farbsynthese, subtraktive 981
Farbwerte, neutrale 983
Farbzuweisungen in CSS 980
Farbzuweisungen in HTML 979
Grundfarben 983
in CSS 1027
nicht druckbare 982
RGB-Dezimalcode (CSS) 980
RGB-Farbmodell 982
RGB-Hexadezimalcode 982
RGB-Hexadezimalcode, verkürzt (CSS) 980
RGB-Prozentangabe (CSS) 980
Vierfarbmodell 981
Windows VGA Farbpalette 983
X11 Farbnamen 982
Fettegrad 1043
Firmen
Apple 304
Netscape 297
NTT DoCoMo 351
Opera 304
TTN 351, 354
Floatlayout 114
Foren
SelfHTML 1172
XHTML Forum 1172
Formatierer 23
Formulare 156
Buttonaufschrift 157

G

Grafiken
Einbindung 146

H

Hersteller
Nokia 1167
Hilfsnavigation 170
Hintergrundfarbe 133
Hintergrundgrafiken 133
Host Language → XHTML Host Language
HTML-Editoren
TopStyle Pro 206
HTML-Versionen
cHTML 31, 350
Doti 1.0 354
HTML 1.0 295
HTML 2.0 25, 295
HTML 3.0 299, 302
HTML 3.2 26, 297
HTML 4.0 26, 303
HTML 4.0 mobile 31, 355
HTML 4.01 26
HTML 4.01 frameset 307
HTML 4.01 strict 305

HTML 4.01 transitional 307
HTML 5.0 304, 346
HTML 5.0, geplante Erweiterungen 347
HTML 5.0, Testimplementierungen 348
HTML Level 0 293
HTML+ (HTMLPlus) 299, 300
iMode 1.0 351
iMode 2.0 352
iMode 3.0 353
iMode 4.0 353
iMode HTML 31
iMode XHTML 31
iMode XHTML 1.0 355
iMode XHTML 1.1 356
mobile Endgeräte 350
Web Applications 1.0 346
WML 2.0 357
XHTML 1.0 28, 308
XHTML 1.0 frameset 310
XHTML 1.0 strict 309
XHTML 1.0 transitional 309
XHTML 1.1 30, 310
XHTML 2.0 340, 359
XHTML 2.0, modulares 342
XHTML Basic 31, 339, 357
XHTML mobile profile 31, 356
XHTML, modulares 312
Hyperlink 24
Defaultpräsentation 143
Hyperlinkcontainer 142, 143
Hyperreferenz
absoluter Pfad 87
Basis-URI 87
Dateiname 87
Pfadangabe 87
relativer Pfad 87
Hypertextsystem 165

I

iMode HTML 351
iMode XHTML 355
Inhaltsbereich 168
Inhaltsgruppe 42
Inhaltsmodell 37, 41
CDATA-Textinhalt 41
Elementinhalt 41
Gemischter Inhalt 41
Inhaltsgruppe 42
Inhaltsgruppe, Kardinalität 42
PCDATA-Textinhalt 41
Schlüsselworte 41
Inlineelemente 137
logische 137
physische 137
präsentative 137
Innenabstand → CSS-Boxproperties, padding
Institution
What WG 952
Institutionen
CERN-Institut 21
IETF (Internet Engineering Task Force) 296
Internet Enginering Task Force (IETF) 25
NILS, www.nils.org.au 280, 282
Open Mobile Alliance 220, 356, 1167
Openwave 1166
Vision Australia 280
W3C, World Wide Web Consortium 299
Wap Forum 1167
Web for all 280, 282
WHAT Working Group 346
World Wide Web Commitee (W3C) 25
Interaction Level Eventhandler 920
Internationalisierung 915
Internet Enginering Task Force (IETF) 25

K

Kommentare
HTML 46
in Stylesheets 102, 1002
Konsistenz 112

L

Language Code 974
Listen
geordnete 150
ungeordnete 150

M

Marke 36
Auslassbarkeit 25, 46
Begrenzer 36
closing delimiter 36
Delimiter 36
Name 36
open delimiter 36
Marker-Attribut → Attribut

Markupsprache 35
Dokumenttyp-Definition 43
MIME-Typ 57
Mobilgeräte 219
Arbeitsspeicher 219
Dateneingabe 220
Displaygröße 219
Displaygrößen 221
Eingabegerät 221
Event-Unterstützung 221
Prozessoren 219, 220
Sonderfunktionen 221

N

Namensraum 52
XHTML 52
Navigation 165
Ad Hoc 167
Globale 167
Grafikgestützte 190
Hierarchische 167
Lokale 167
mit Listen 173
mit Textabsätzen 172
mit Zeilenumbrüchen 171
ungeordnete Liste 173
Navigationsgestaltung 166

O

open delimiter 36
Open Mobile Alliance 220, 356

P

Parameter-Entity 45
Parser 23
Zwischencode 23
PCDATA-Textinhalt 42
Positionierung
absolute 126
Bezugskoordinatensystem 124
feste 126
relative 126, 127
statische 126
Positionslayout 122
Programmiersprache 35

Q

Quelltext
Reorganisation 120
Quelltextreihenfolge 119

R

Radiobuttons 157
Ruby 30, 311
Ruby-Spezifikation 312

S

Schema-Locator 83
Screenreader 243
Servicebereich 168
SGML 25
SGML-konforme Deklaration 44
SGML-Vereinbarung
leere 47
Skiplinks → Sprunglinks
Small Screen Rendering Modus 233
Sprunglinks 120
Standardattribute 915
Accessibility (IE) 947
accesskey (IE) 948
atomicselection (IE) 951
class 916
contenteditable (IE) 952
Databinding (IE) 947, 949
datafld (IE) 950
dataformatas (IE) 950
datapagesize (IE) 951
datasrc (IE) 949
dir 916
Editing (IE) 947, 951
Eventhandler 919
hidefocus (IE) 948
id 917
lang 917
onblur 921
onchange 921
onclick 922
ondblclick 923
onfocus 923
onkeydown 923
onkeypress 924
onkeyup 924
onload 924
onmousedown 925

onmousemove 925
onmouseout 925
onmouseover 926
onmouseup 926
onreset 926
onselect 927
onsubmit 927
onunload 927
style 918
tabindex (IE) 949
title 918
unselectable (IE) 953
xml:lang 919
Startmarke 22, 36
Starttag → Startmarke
Steuerzeichen
Maskierung 40
Stylesprachen
DSSSL lite 302
DSSSL Online 302

T

Tabellen 151
Texteinzug 136
Textinhalt
CDATA 41
PCDATA 41
Tools
Absolute Color Picker 286
Accessibility Tools 282
Accessibility-CSS 1171
Accessibility-Toolbar 1171
Bildschirmlineale 283
Browser-Plug-ins 287
Color Contrast Analyzer 244, 248, 282
Color Spy 285
Colorblind Web Page Filter 1171
ColorZilla (Firefox) 284
Delorie Lynx Viewer 291
Developer Toolbars 276
DOM-Inspector (Mozilla) 287
Farbkontrast Analyzer 282
Firebug (Firefox) 288
HTML Tidy 196
HTML Tidy, accessibility-check 204
HTML Tidy, add-xml-decl 203
HTML Tidy, Batch-Verarbeitung 205
HTML Tidy, char-encoding 204
HTML Tidy, Clean 200
HTML Tidy, Dateiausgabe 199
HTML Tidy, Dateimodifikation 200
HTML Tidy, doctype 204
HTML Tidy, drop-empty-paras 202
HTML Tidy, drop-font-tags 202
HTML Tidy, enclose-text 202
HTML Tidy, GUI-Versionen 206
HTML Tidy, Ingegration in HTML-Kit 210
HTML Tidy, Kommandozeile 197
HTML Tidy, Konfigurationsdatei 205
HTML Tidy, logical-emphasis 203
HTML Tidy, lower-literals 203
HTML Tidy, Optionen 201
HTML Tidy, output-xhtml 203
HTML Tidy, quote-ampersand 203
HTML Tidy, quote-marks 203
HTML Tidy, Stapelverarbeitung 205
HTML Tidy, Tidy GUI (Windows) 206, 208
HTML Tidy, Tidy UI (Windows) 206
HTML Tidy, write-back 202
HTML Tidy, XHTML-Ausgabe 199
Inspect Element (Mozilla) 287
InspectThis (Mozilla) 287
Internet Explorer Developer Toolbar 278
Measure It (Firefox) 283
Openwave Phone Simulator 236
Styleswitcher 1171
Wave Toolbar 281
Web Accessibility Toolbar 244, 280, 291
Web Developer Toolbar 276
Webdeveloper-Toolbar 1170
Yellowpipe Lynx Viewer (Firefox) 290
Yellowpipe Lynx Viewer (IE) 290

U

Überschriftenhierarchie 134
Universalattribut 90
Universalattribute → Standardattribute
URI-String 971
Bestandteile, funktionale 972
Portbezeichner 972
Querystring 972
Zeichen, verbotene 973

V

Validatoren 1169
Validome HTML Validator 1170
W3C CSS-Validator 1169

W3C Feed Validator 1169
W3C HTML Tidy 1170
W3C Link Checker 1170
W3C XHTML-Validator 1169
WDG HTML Validator 1169
Validierung 59
Verkürzungsvereinbarung 46

W

Web Accessibility Initiative (WAI) 241
Web Applications 304, 347
Web Content Accessibility Guidelines (WCAG 241
Web Controls 347
Web Forms 347
Webbrowser 23
Webzines
A List Apart 1172
CSS Zen Garden 1172
Werte und Maßeinheiten 1023
Wurzelelement 23, 52
XHTML 51

X

XForms 359
Aufbau eines XFormulars 375
Auswahlfelder 380
Auswahlfelder, Einfachauswahl 380
Auswahlfelder, Mehrfachauswahl 380
CSS-Selektoren 374
CSS-Selektoren, CSS3 374
CSS-Selektoren, XForms 374
Datentypen 368
Datentypen, eigene 368
Datentypen, XML-Schema 368
Datumsfeld 396
Elemente 361
Entwicklungsumgebungen 361
Entwicklungsumgebungen, XFormation 361
erweitern 365
Events 371
Events, Error Indications 372
Events, Eventdefinitionen 372
Events, Initialization Events 372
Events, Interaction Events 372
Events, Notification Events 372
Funktionen 369
Funktionen, Auswertungskontext 370
Funktionen, Boolesche 370
Funktionen, Datum und Zeit 371
Funktionen, DOM-Manipulation 369
Funktionen, Nodesets 371
Funktionen, Propertyermittlung 371
Funktionen, Zahlenwerte 370
Hints 378
Implementierungen 360
Implementierungen, Chiba Project 361
Implementierungen, clientseitige 360
Implementierungen, FormFaces 360
Implementierungen, serverseitige 361
Implementierungen, X-Smiles 360
Mehrfaches Submit 385
Module 362
Module, Action Module 364
Module, Core Module 362
Module, Extension Module 365
Module, Form Controls Module 363
Module, Group Module 366
Module, MustUnderstand Module 366
Module, Repeat Module 367
Module, Switch Module 367
Namensraum 362
Range 394
Reglementierung durch XML-Schema 379
Übertragungsmethoden 384
Vorgabewerte 377
XPath 368
XFrames 64, 359, 399
Aufruf eines XFrames-Dokuments 406
Elemente 399
Elemente, frame 404
Elemente, frames 400
Elemente, group 402
Elemente, head 400
Elemente, style 401
Elemente, title 401
XHTML
Host language 31
Kleinschreibung 51
modulares 30
XHTML 2.0-Attributmodule
Bi-Directional 345
Core 344
Edit 345
Embedding 345
Events 345
Hypertext 345
I18N 344
Image Map 345
Media 345
Metainformation 345

Role 345
Style 345
XForms 345
XHTML 2.0-Attributsammlungen
Common 344
XHTML 2.0-Module
Core Module 342
Document Modul 342
extern definierte 344
Handler Modul 343
Hypertext Modul 343
Image Modul 343
List Modul 343
Metainformation Modul 343
Object Modul 343
Role Access Modul 343
Ruby Modul 344
Structural Modul 343
Style Sheet Modul 343
Tables Modul 344
Text Modul 343
XForms Modul 344
XML Events Modul 344
XHTML Host Language 315
XHTML Basic 339
XHTML Modularization 312
XHTML-Attributsammlungen
Attributsammlungen, leere 314
Common 313
Core 313
Events 314
I18N 313
Style 314
XHTML-Dokument
streng konformes 59
XHTML-Module
Applet Modul 328
Base Modul 328
Basic Forms Modul 321
Basic Forms Modul, Attribute 322
Basic Tables Modul 323
Basic Tables Modul, Attribute 324
Bi-Directional Text Modul 328
Client Side Image Map Modul, Attribute 329
Client-Side Image Map Modul 329
Core-Module 314
Edit Modul 330
Formularmodul 325
Frames Modul 330
Hypertextmodul 318
IFrame Modul 331
Imagemodul 327
Intrinsic Events Modul 331
Konformität 315
Konformität, Host Language 315
Konformität, Integration Set 315
Legacy Modul 332
Legacy Modul, Attribute 333
Link Modul 335
Listenmodul 319
Meta Information Modul 335
Name Identification Modul 336
Object Modul 336
Präsentationsmodul 320
Ruby-Modul 311
Scripting Modul 336
Server Side Image Map Modul 337
Strukturmodul 316
Style Attribut Modul 337
Style Attribut Modul, Ausnahmen 337
Style Sheet Modul 338
Tabellenmodul 326
Target Modul 338
Target Modul, benötigte Zusatzmodule 338
Target Modul, betroffene Elemente 338
Textmodul 316
XHTML-Namensraum 52
Namensraumstring 52
XML-Events 372
XML-konform 58
XML-konforme Deklaration 48
XML-Schema 83

Z

Zeilenabstand 136
Zeilenkontext 116
Zwischencode 23